DEVELOPMENT AND EVALUATION OF POSITIONING SYSTEMS FOR AUTONOMOUS VEHICLE NAVIGATION

By

ROMMEL E. MANDAPAT

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2001

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2001		**00-00-2001 to 00-00-2001**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Development and Evaluation of Positioning Systems for Autonomous Vehicle Navigation	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Center for Intelligent Machines and Robotics,Department of Mechanical Engineering,University of Florida,Gainesville,FL,32611**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**276**	

ACKNOWLEDGMENTS

The author would like to express his deepest gratitude to the members of his supervisory committee, Dr. Carl Crane III, Dr. Joseph Duffy and Dr. Paul Mason for their leadership and guidance and for the opportunity to participate and contribute to an exceptional graduate program.

Special thanks go to David Armstrong, Dave Novick and Jeff Wit for their active participation and continuous help in the development of this research project. Thanks also go out to the people of CIMAR, most notably those directly involved in the autonomous vehicle project for their help, suggestions, and friendship.

The author would also like to recognize the continued support and funding given by Wright Research Laboratory at Tyndall Air Force Base in Panama City, Florida as well as the individual contributions of their staff to this project.

Sincerest thanks go out to the author's parents, Gil and Cecilia Mandapat for their undying support and encouragement.

Finally, the author wishes to send out his most heartfelt gratitude and love to his wife, Eia, for supporting and inspiring him to make the most out of this opportunity.

TABLE OF CONTENTS

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

DEVELOPMENT AND EVALUATION OF POSITIONING SYSTEMS
OR AUTONOMOUS VEHICLE NAVIGATION

By

Rommel E. Mandapat

December 2001

Chairperson of the Supervisory Committee: Professor Carl D. Crane III

Major Department : Mechanical Engineering

Successful navigation of an autonomous vehicle is made possible by accurately measuring real-time position and orientation. Positioning systems should provide fast, accurate and reliable operation through varying conditions.

The current positioning system being used on Center for Intelligent Machines And Robotics' (CIMAR) Navigation Test Vehicle (NTV) consists of an Inertial Navigation System (INS) and Global Positioning System (GPS) integrated through an external Kalman Filter (KF). The INS component is a Honeywell H-726 Modular Azimuth Positioning System (MAPS) and the GPS component is an Ashtech Z-12 Differential GPS. The high data rate of the INS directly complements the high accuracy of the GPS. System performance shows sustained positional accuracy of less than ten centimeters at output data rates of up to 10 Hz. The high system cost, however, limits its versatility in application. The focus of this thesis is to explore two alternative low-cost positioning systems.

A second positioning system is the Novatel Beeline GPS, which consists of dual antennas to measure both vehicle position and dual-axis orientation. The drawbacks of the system are inherent to GPS, including satellite visibility at all times and slow recovery after data loss. This stand-alone system features position data to within 20 centimeters at data rates up to 5 Hz.

A third system is a low-cost INS/GPS integrated positioning system. It makes use of a Honeywell HG1700AG11 Inertial Measurement Unit (IMU) and a Novatel RT-20 Differential GPS. Raw acceleration and angular velocity data at 100 Hz from the IMU is averaged down to 12.5 Hz for processing by the Navigation Processor. The Navigation Processor performs a two-stage alignment to determine the IMU's initial orientation angles. The second alignment stage uses a Kalman Filter (KF) to reduce platform tilt errors. Once aligned, data is used to plot out a navigation solution to provide position, velocity and orientation. To reduce inherent IMU drift, integration with the GPS is done through a secondary KF, which overcomes GPS data loss and rejects DGPS data with an error over 1 meter. Under ideal conditions, overall system performance is expected to reach twenty-centimeter positional accuracy at 12.5 Hz.

The performance of these three systems are evaluated and compared using the similar test set-ups, simulating ideal and adverse operating conditions. Overall, the MAPS/GPS shows a slower drift rate and better recovery time. However, the IMU/GPS compares favorably in that its adequate performance and low cost makes it ideal for autonomous ground vehicle navigation.

CHAPTER 1
INTRODUCTION

Project Background

In 1991, the Air Force Research Laboratory (AFRL) at Tyndall Air Force Base, initiated a program dealing with autonomous vehicle systems that could be applied to a variety of Air Force needs

Autonomous navigation is the focus of work conducted at the University of Florida's Center for Intelligent Machines and Robotics (CIMAR). The navigation task can be further subdivided into sub-tasks including vehicle control, path planning, vehicle positioning, path following, and obstacle avoidance.

In order to accomplish these tasks, CIMAR developed its own Navigation Test Vehicle (NTV, see Figure 1.1), a retrofitted Kawasaki MULE 500. Initially, the NTV was equipped with a VME computer running under the VxWorks operating system. To have closed-loop control over the NTV's functions, actuators and encoders have been integrated into the steering, throttle, brake and shifter mechanisms. Once fully developed and tested on the NTV, the technology is transferred to other Air Force systems.

One application of the autonomous vehicle technologies was the cleanup of various DOD facilities from unexploded ordnance (UXO). Autonomous vehicle operation removes human operator hazards and ensures high site search efficiency.

Figure 1.1: Navigation Test Vehicle Kawasaki Mule 500

The cleanup of buried munitions requires two steps, survey and cleanup. The survey involves sweeping 100% of the area using an Autonomous Survey Vehicle (ASV). The ASV is a modified John Deere Gator (shown in Figure 1.2) towing a sensor package composed of a magnetometer array and a ground penetrating radar array. As the ASV navigates, it collects and stores time-tagged position and sensor data. This data is then post-processed to determine the locations of possible UXO. After the survey step, the cleanup involves removal of buried ordnance. This is accomplished by an autonomous John Deere excavator shown in Figure 1.3.

As part of a mission to clear a 50 x 50 yard beachhead area of mines and other obstructions, a D7G bulldozer (see Figure 1.4) outfitted with a mine plow was also automated using NTV technology. This automated bulldozer's job was to assist in the deployment of the Marines and their supplies.

Figure 1.2: Autonomous Survey Vehicle (John Deere Gator).

Figure 1.3: John Deere Excavator.

Figure 1.4: Joint Amphibious Mine Countermeasures D7G Dozer.

The next generation of autonomous vehicles to utilize NTV technology is built on a commercially-available vehicle built by ASV. The All-Purpose Remote Transport Vehicle

(ARTS), shown in Figure 1.5 includes a front lift assembly and tele-remote package developed by Applied Research Associates, Inc. (ARA). Another version has a manipulator arm with a gripper on one side and a high-pressure waterjet attachment on the other. In October of 1999, as part of a demonstration for the Joint Architecture for Unmanned Ground Systems (JAUGS), the ARTS swept an area, dropping discs from its rear mine-laying simulating mechanism. The NTV, towing the magnetometer and ground penetrating radar trailer, swept the same area locating the discs for eventual clean-up. JAUGS is being developed by the Department of Defense Joint Robotics Program.

As part of the ongoing effort to standardize the interface of the autonomous ground vehicle systems, the first vehicle to be fully JAUGS-compliant is the AMRADS (Autonomous Mobility Research and Development System, see Figure 1.6). Its main purpose is similar to the NTV in that it serves as a testbed for autonomous navigation systems applied on a tracked ground vehicle. During a demonstration of cooperative vehicle systems in May of 2001, the AMRADS performed perimeter surveillance while the NTV acted as a autonomous re-supply vehicle as well as ambulance (see Figure 1.7).

Figure 1.5: All-Purpose Remote Transport (ARTS) Vehicle.

Figure 1.6: Autonomous Mobility Research and Development System (AMRADS).

Figure 1.7: Cooperative Vehicle Demonstration (AFRL, May 2001).

Current NTV Technology

The Navigation Test Vehicle (NTV) has served as the test-bed for all the technology developed for the autonomous vehicle project. There are currently seven graduate students and two undergraduate students involved in developing systems to further the application and successful implementation of autonomous ground vehicle navigation systems. Aside from military use, the NTV has also been utilized to investigate automating a variety of applications such as golf course grass mowing (Figure 1.2) and orange grove inspection (Figure 1.8).

Figure 1.8: NTV Navigating Orange Grove.

Navigation Processes

Vehicle Control, an essential primary step, involves automating all vehicle operational functions effectively switching them from manual to either remote operation or to a fully autonomous mode. Currently, the NTV employs DC servo motors on both the steering and throttle. An encoder accurately measures the steering angle. Linear actuators are used for the brake and shifter.

The next navigational sub-task is Path Planning. There are 2 types of paths: Go To Goal, and Sweep The Field. 'Go To Goal' involves efficiently driving from a starting location to a goal location. This may either be done through the shortest path possible or through existing roads previously mapped out. In 'Sweep the Field', the NTV picks four corner points demarcating the field, and sweeps it insuring 100% coverage. An off-line path planner is used to generate these paths [Ran94a]. When generating these paths, the off-line path planner takes into account any known obstacle by representing them as polygons in an area map [Wit96]. The shortest path around all obstacles is calculated by using the A* search algorithm [Jar83].

The next task is to accurately determine vehicle position. This will allow the vehicle to effectively follow the planned path. The NTV's current positioning system is an integrated MAPS Inertial Navigation System / Global Positioning System (MAPS/GPS) capable of providing high accuracy position, velocity and orientation data at high rates. The details of the MAPS/GPS positioning system is discussed in detail in Chapter 3.

Once position has been established, the vehicle has to perform path following or path tracking. This involves continually generating the vehicle's desired speed and steering angle to keep the vehicle on the desired path. By choosing the center of the vehicle's rear axle as the point to control, it has been shown that control of the vehicle's speed and heading are kinematically decoupled [Shi92]. The decoupling permits independent calculation of the desired speed and steering angle. Vehicle speed is set at a constant three miles per hour (1.34 m/sec). The desired steering angle is continuously recalculated geometrically by comparing the present position of the vehicle to the planned path. The planned path consists of subgoals which the vehicle constantly tries to reach. Once the vehicle is within a predetermined distance, it looks to the next subgoal until it reaches the final goal [Ran94b].

And the last step in navigation involves obstacle avoidance. This is important in adapting to unknown or changing conditions that are not indicated in the planned path. In order to avoid obstacles along the path, the NTV must first detect them and then efficiently avoid them. Current sensors used for obstacle avoidance on the NTV include a Polaroid ultrasonic transducer array and a Sick Laser Range Detector (Figure 1.9). The ultrasonic sensors give wider sensing coverage (180 degree lateral coverage and a 60 degree top to bottom on the vehicle front side). The laser provides longer detection range (200 feet in ideal conditions).

Once an obstacle is detected, the vehicle slows down and tries to safely steer around the obstacle.

Figure 1.9: Polaroid Ultrasonic Transducer Array and Sick Laser Range Detector.

Modular Architecture eXperimental (MAX)

In order to improve the versatility and functionality of the NTV, CIMAR designed a modular system architecture that allows interchangability of the individual components. By standardizing the interface between system components, the Modular Architecture eXperimental (MAX) allows for rapid system reconfiguration. The modular structure of MAX is depicted in Figure 1.10. The key of the MAX Architecture lies in the standard message sets defined for each component. MAX shortens the system development time as individual components can be built and tested using essentially the same NTV set-up. Furthermore, component hardware design becomes non-vehicle specific, which gives increased design flexibility. Finally, an important advantage of the modular architecture is the relative ease of adapting it to automate other vehicles. In July of 1999, CIMAR and AFRL together with the companies Applied

Research Associates and Wintec, used the MAX architecture to full advantage by automating an ARTS in six weeks. This vehicle would have normally taken six months to automate.

Implementation of MAX is further enhanced by standardizing system software. Presently, all systems run C/C++ code using the Lynx Real Time Operating System. Systems share libraries as well as the same software structure and architecture. Each system also utilizes shared memory to allow software blocks to reuse essential data efficiently. The downsides to the architecture include generally larger hardware space requirement and slightly higher overall hardware cost. One of the immediate goals is to convert the NTV system software to run under Red Hat Linux 7.0 Operating System. Also, a Message Routing System (MRS) has been designed to optimize data handling within the different systems at every level. The MRS will operate through an ethernet connection.

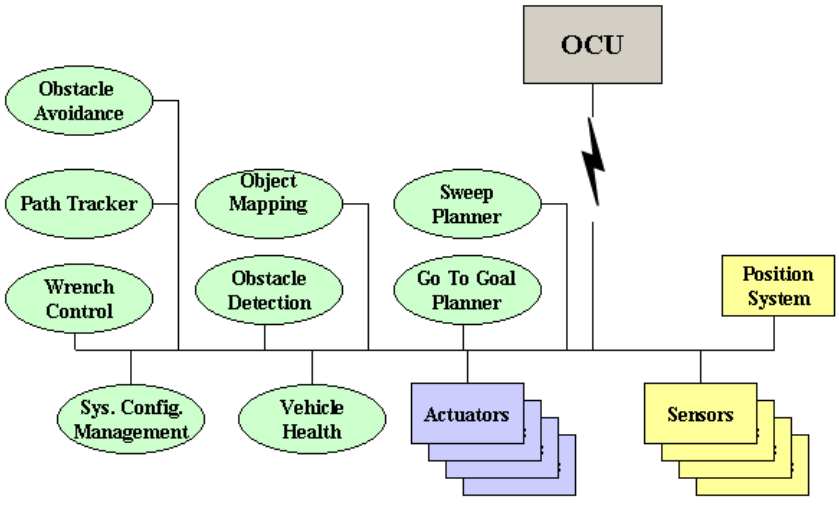


Figure 1.10: Current MAX System Structure.

Currently, the Navigation Test Vehicle uses a gas engine for propulsion and a gas generator to power all the electronic components in its rear cabinet, with an UPS as back-up. A laptop allows one to monitor the system output and make changes on the fly. The NTV is also capable of operating under Tele-remote operation and controlled through an Operator Control Unit (OCU – Silicon Graphics workstation).

The NTV rear cabinet houses all the individual system shelves (see Figure 1.11). Each shelf contains a fully independent system component which can be replaced without having to modify the other system component settings.

On a larger scale, AFRL and CIMAR are involved in the development of a standard approach in the design and specification of autonomous vehicles being developed by military contractors and research groups. This DOD thrust is called the Joint Architecture for Unmanned Ground Systems (JAUGS). Once fully developed and implemented, there will be greater cooperation and advancement in the field of Autonomous Ground Vehicles (AGV).

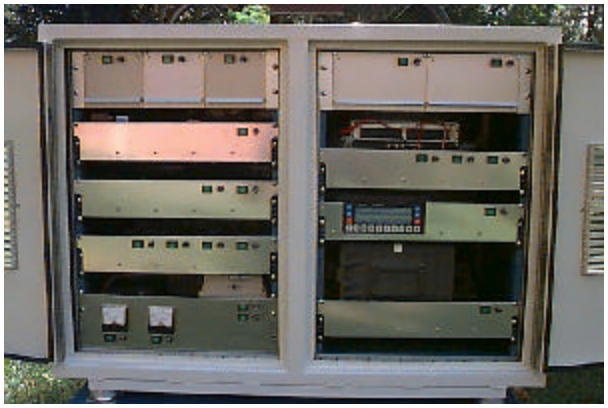


Figure 1.11: NTV Rear Cabinet.

Current CIMAR Research

The main areas of research currently tackled at CIMAR include obstacle detection mapping, backing with a trailer, the development of a portable operator control unit, the coordinated control of multiple vehicle systems, the development of a beacon based positioning system, and the integration of stereo vision sensing systems. In the Detection Mapping System, a local grid map relative to the vehicle is built using obstacle sensor data. As the vehicle navigates, the map is continuously updated thereby allowing it to traverse and avoid obstacles efficiently. A vehicle control system is being developed to successfully drive the NTV with a trailer in reverse. The trailer hitch provides closed-loop feedback control allowing the NTV to follow a path in reverse accurately at relatively high speeds. The Portable Operator Control Unit being developed will replace the current Silicon Graphics workstation set-up with a laptop running under the Linux Operating System and will be able to control the NTV in full autonomous or tele-remote operation mode.

Among the recently started research efforts is the coordinated control of a multiple vehicle system. The NTV acts as the main or "mother" vehicle with several smaller "child" vehicles which may act as autonomous sensor units. These "child" vehicles (Figure 1.12), once deployed, can form a sensor array covering a wider area of the field to be surveyed. This kind of task division is perfectly suited for buried munitions detection. The data from the "child" vehicles can be sent back to the "mother" unit through RF. Also, the "mother" vehicle knows its absolute position while the "child" vehicles know its relative position to the "mother" vehicle. Another type of beacon-based positioning system being explored uses laser light to track specific markers laid out around the perimeter of the field.

Figure 1.12: Eliminator ("Child" vehicle).

And lastly, three-dimensional obstacle detection and mapping is the long-term objective of the Detection and Mapping System. A stereovision camera system is to be integrated into the system. Using dual cameras, exact 3-D geometry and location of obstacles can be accurately measured and stored. Currently, a single Sony CCD camera is used to perform edge following, allowing the NTV to navigate through a marked path.

Thesis Focus

The current positioning system on the NTV integrates a Honeywell H-726 Modular Azimuth Positioning System (MAPS) with an Ashtech Z-12 differential GPS through an external Kalman Filter (KF). This system provides high accuracy position (less than 6-7 centimeter error) and orientation (0.01 degree error) at high rates (12.5 Hz). However, the high cost of the system ($150,000) has made it unsuitable for certain applications such as multiple vehicle systems.

The focus of this thesis is to investigate two lower-cost alternative positioning systems, the Novatel Beeline RT-20 GPS and the Honeywell IMU / Novatel GPS integrated system.

Each system is developed, integrated into the NTV, tested and benchmarked against the MAPS-Pos system. Additional tests are performed using the AMRADS as the vehicle platform. Finally, the suitability of each alternative system is discussed.

CHAPTER 2
RELATED RESEARCH

The process of following a path on a map through predetermined latitudes and longitudes is called navigation, and the process of pointing the vehicle to follow a chosen path is called guidance [Law93]. Since a vehicle cannot effectively get to a desired location without knowing its current as well as its past location, positioning systems have been essential components in autonomous vehicle navigation.

Types of Positioning Systems

The type of positioning system defines the type of application it is most suited for. There are two general types of positioning systems: dead reckoning systems and active beacon systems.

Dead Reckoning Systems

Dead reckoning involves continuously measuring the heading and distance traveled in order to calculate position [Wit96]. Dead reckoning offers a low cost solution to vehicle positioning. However, the main characteristic of this type of positioning system is that the positional accuracy drifts or degrades over time. This is due to the inherent sensor error characteristics which accumulate over time. A common example of a dead reckoning instrument is an encoder which can be used to calculate the distance traveled by the vehicle

wheels. And to calculate the vehicle orientation, heading sensors such as a compass can be used together with the encoders.

A more accurate and versatile dead reckoning instrument is the Inertial Measurement Unit (IMU). An IMU is basically composed of three accelerometers and three gyroscopes mounted on orthogonal axes. These will be discussed later in more detail.

Active Beacon Type Systems

Active Beacon systems use either radio frequency, ultrasonic or light sources to transmit relative positioning between the vehicle and known reference points. The principle behind calculating vehicle position lies in the concept of triangulation and trilateration. In triangulation, a rotating receiver on the vehicle calculates its position relative to three or more field markers. Using the angles between the vehicle's longitudinal axis and each marker, the exact vehicle position can be calculated. In trilateration, the distance between the vehicle receiver and the markers are used instead. Some systems have the active beacon rotating on the vehicle while the markers passively reflect the signal back to the vehicle.

The main advantage of an active beacon system is the positional accuracy does not drift over time. The systems are also computational efficient requiring less processing power. The disadvantage lies in the inflexibility to the changing survey environment since the field markers have to be set in place and surveyed beforehand. Also the range of survey is limited to the area bound by the field markers.

An example of a commercially available system is the Geodimeter 600 ATS (see Figure 2.1), which is typically used for land surveying vehicle positioning. The typical specifications are listed in Table 2.1.

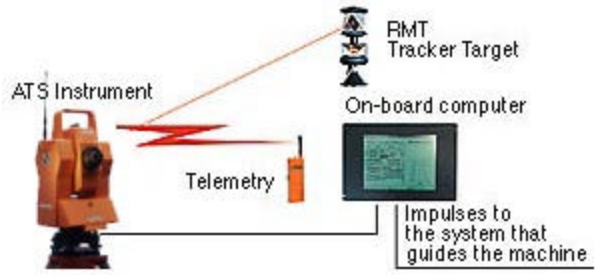


Figure 2.1: Geodimeter 600 ATS System Set-Up .

Table 2.1: Geodimeter 600 ATS System Specifications.

Parameter	Specifications
Accuracy at constant speed (1 m/s at 300m)	
Horizontal	±2 mm + 14 ppm
Vertical	±2 mm + 14 ppm
Slope distance	±2 mm + 14 ppm
Timing of data	
Rate 1-6 Hz user definable	±1 ms (one meas, each 166 ms)
Latency	183 ms (incl. Radio modem)
Synchronization of measurement data	<5 ms
Maximum Acceleration of Target on short distance	
Radial acceleration	10 grads/s2, 9 deg/s2
Maximum Velocity of Target	
Radial speed	25 grad/s, 23 deg/s
Axial speed	6 m/s

Inertial Measurement Unit

An Inertial Measurement Unit (IMU) is a group of inertial sensors (3 gyros and 3 acceleremoters) that measure vehicle orientation and acceleration. There are 2 types of IMUs, the inertial platform (or gimbaled type) and the strapdown system.

An Inertial Platform (Figure 2.2) uses gyros to maintain the accelerometers in a fixed attitude, i.e., the gimbaled platform serves to define the directions for the measurements of

acceleration. The platform is controlled by an electric torque motor. This together with the gimbals, make the inertial platform mechanically complicated.

To simplify the mechanical construction of the IMU, the strapdown system emerged. The strapdown system (Figure 2.3) replaces the gimbals with a computer that simulates their presence electronically. Here, the gyroscopes and accelerometers are rigidly connected to the vehicle structure so that they move with the vehicle. They are mounted along mutually-orthogonal axes, with the gyros measuring change in angle about the roll, pitch and yaw axes, while the accelerometers measure accelerations along the roll, pitch and yaw axes. Unlike the inertial platform, the strapdown gyros measure the angles turned, up to the maximum rotation rate expected. Table 2.2 is a comparison of the gimbaled platform and strapdown type IMU [Law93].

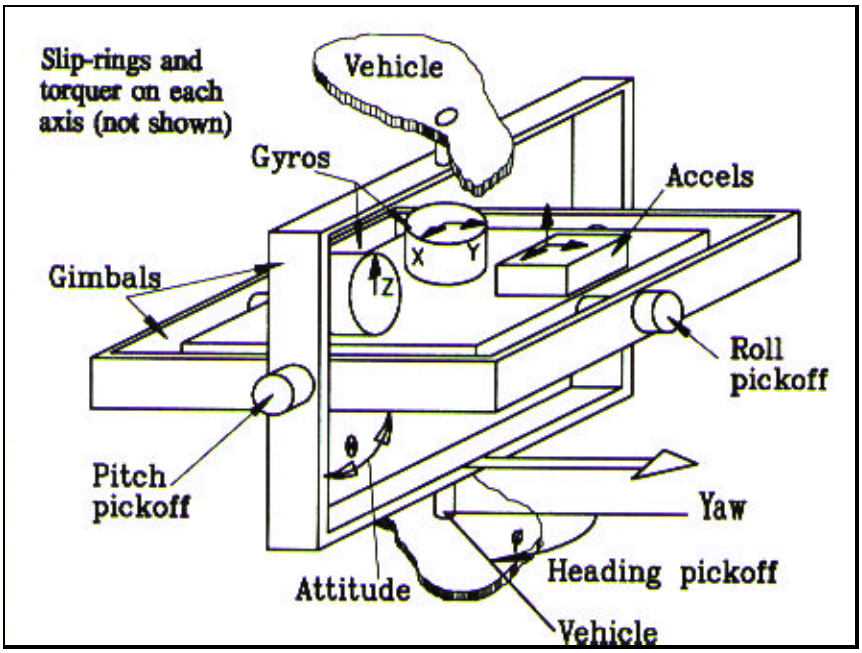


Figure 2.2: Gimbaled Platform Type IMU.

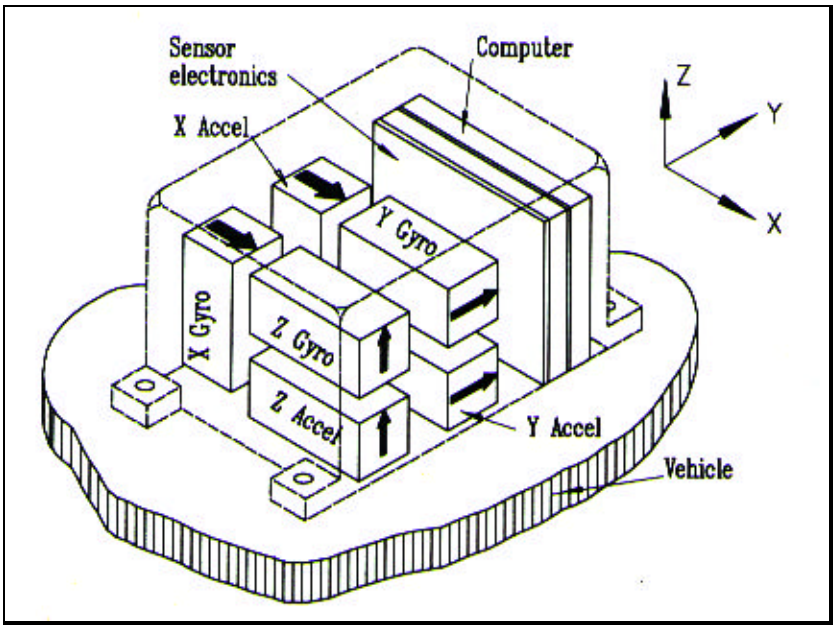


Figure 2.3: Strapdown IMU.

Table 2.2: Comparison of Gimbaled Platform versus Strapdown Type IMU.

Gimbaled Platform	Strapdown
Advantages	
1) they have lower errors than strapdown systems 2) null gyro operation eliminates anisoinertia and angular acceleration errors 3) they can operate with vehicle rotation rates over 1000 deg/s 4) they self-align by gyrocompassing 5) they can calibrate the sensors by platform rotations 6) gyro torquer errors do not leas to attitude error	1) they are lighter, simpler and cheaper 2) they are easily configured for odd-shaped spaces 3) they have high reliability due to no moving parts
Disadvantages	
1) they are mechanically complicated 2) they are larger and more expensive 3) they are less reliable than strapdown systems	1) they have low maximum rate (400-600 deg/s) 2) they are difficult to align 3) the sensors cannot be easily calibrated so must be stable 4) system rotation induces sensor errors 5) their accelerometer bias errors accumulate

The strapdown Inertial Measurement Unit is a raw sensor that outputs delta angles and delta velocities. Such data is then passed to a Navigation Processor, which is essentially a computer that calculates a dead-reckoning navigation solution with the use of navigation equations run through a Kalman filter. The Navigation Processor then outputs the position and orientation of the IMU, normally in local vehicle coordinates (latitude, longitude, altitude, roll, pitch, yaw). An IMU with a navigation processor and support electronics is called an Inertial Navigation System (INS). An example of this is the Honeywell H-726 MAPS.

Eglin Air Force Base together with Charles Stark Draper Laboratory, has developed, fabricated and tested the first three-axis microelectromechanical system (MEMs) ever assembled utilizing three MEM gyros. Each of the hybrid MEM gyros is on a single printed circuit board that is fixed to a conservative 5-in cube layout for maximum access and testability [Bie99].

Efforts to build the Advanced Tactical Inertial Measurement Unit (ATIMU) are aimed at developing a miniature (15 in^3), extremely low-cost (less than $2000) tactical missile grade IMU using innovative MEMs technology. This next generation IMU will have a gyroscope drift rate performance requirement of 1 deg/h and an accelerometer bias stability of 300 ug [Bie99].

The Fiber Optic Gyro (FOG) IMU is a small (25 in^3), low-cost (less than $6000) tactical missile grade IMU using interferometric FOG technology and integrated silicon accelerometers (ISA) [Bie99].

The high data output rates of the IMU and INS make it ideal for fast navigation. However, the inherent drift of the inertial sensors necessitate the aiding by GPS to maintain accurate position and orientation at all times.

Global Positioning System

The Global Positioning System (GPS), maintained by the Department of Defense consists of 24 geosynchronous satellites, or space vehicles (SVs) which orbit the earth. These SVs transmit data on two microwave carrier signals, L1 frequency (1575.42 MHz) and L2 frequency (1227.60 MHz). The L1 carries navigation message and the Standard Positioning System (SPS) code signals. The L2 carrier is used to measure ionospheric delay by Precise Positioning System (PPS) equipped receivers. The SVs make up the space segment of GPS.

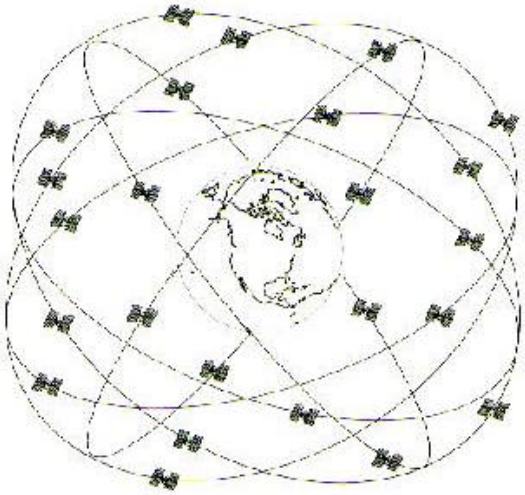


Figure 2.4: NAVSTAR Satellite Orbit Arrangment.

The control segment of GPS consists of tracking stations located throughout the world. These control facilities monitor signals from the SVs and calculate precise orbital data and SV clock corrections for each satellite. The Master Control facility (located at Falcon Air Force Base in Colorado) then uploads these corrections to the satellites.

The user segment of GPS consists of GPS receivers. Four GPS satellites are required to calculate position and time. Using PPS allows the user about 18 meter horizontal accuracy,

28 meter vertical accuracy and 100 nanosecond time accuracy. SPS, which has been available to all users worldwide receivers, was initially degraded the DOD through the use of Selective Availability (SA), which introduces a time-varying bias. However, in May of 2000, SA was turned-off and even L1 receivers can reach PPS accuracy levels.

Accuracy is greatly affected by bias errors, which include tropospheric delays, ionospheric delays, satellite clock and ephemeris data, multipath and SA. Multipath occurs when the signal from a satellite arrives from more than one propagation route (reflections). The biggest error source is SA. Afterwhich, the rest of the bias errors can be eliminated by using differential GPS (DGPS). The basic principle behind DGPS is to place a receiver (base) at a known location to calculate these bias errors for each satellite.

The errors are then transmitted via radio modem to a remote receiver. The remote receiver uses the corrections to calculate its position more accurately (generally to within about a meter). In order for the principle to work, the base and the remote receivers must see the same bias errors (or same satellites), which requires them to be relatively close (within 12 miles).

To improve the accuracy even further (< 1.0 m), a differential carrier phase tracking technique must be used. Here, the phase shift between the same signal measured at 2 different locations translates to the relative distance of the 2 receivers. An example of such is the Ashtech Z-12 receiver currently on the NTV. Also, it is important to note that carrier-phase allows for Real-Time Kinematic (RTK) measurements which is critical in precise navigation.

A variation of DGPS involves differential corrections broadcast on radiobeacon transmissions. These allow for a single remote receiver capable to reading DGPS beacon

signals to calculate position to within a meter. This also eliminates the need for a base station, which may compromise the range of the remote receiver.

Problems associated with GPS include direct line-of-sight with the satellites and slow update rates. GPS is highly susceptible to signal loss when the receiver does not have clear direct line-of-sight with the satellites. Thus, operating in close proximity to natural obstructions such as trees and mountains, as well buildings can seriously compromise its performance. Recently, receivers have been developed to operate under dense foliage. Also, data output rates are normally 1 Hz with some receivers able to handle position update rates up to 10 Hz.

The tremendous acceptance level of GPS, for positioning and navigation purposes, comes because of the extreme ease of use, the relatively high-accuracy level, and the excellent reliability factor. The system's operational simplicity, accuracy, and reliability are the keys, which make it an indispensable military resource.

Military applications include enroute navigation, terminal navigation, low-level navigation, non-precision approach, target acquisition, close air support, missile guidance, command and control, air drops, surveying and mapping, time synchronization, rendezvous, bombing, drone vehicle control, base/site preparation, search and rescue, reconnaissance, mine placement, space navigation. During Operation Desert Storm, GPS was used extensively for land navigation in the desert, troop movements and logistical support [Cla96].

Table 2.3, 2.4, 2.5 and 2.6 outline the specifications of several commercially available GPS receivers.

Table 2.3: Novatel GPS Receivers.

	RT-2	RT-20
Position Accuracy		
standalone (SA off)	11 m CEP*	15 m CEP*
differential (RT-2)	1 cm + 2 ppm*	0.20 m CEP*
post-processed	± 5 mm + 1 ppm*	± 5 mm + 1 ppm*
Time to first fix (cold start)	67 s	67 s
Reacquisition (warm start)	1 s L1, 10 s L2	1 s L1, 10 s L2
Data Rates		
raw data	10 Hz	20 Hz
position	10 Hz	10 Hz (5 Hz RT-20)
Time accuracy (SA off)	102 ns RMS*	50 ns RMS*
Channel Tracking	12 L1 / 12 L2 + carrier tracking	12 L1 + carrier tracking
Power Consumption	9.75 W	8 W
Physical Dimensions		
Millenium (card)	17.9 x 10 x 1.5 cm	16.7 x 10 x 1.5 cm
PowerPack-II	21 x 11.1 x 4.7 cm	20.8 x 11.1 x 4.7 cm
ProPak-II (Beeline)	25.1 x 13.0 x 6.2 cm	24.5 x 13.0 x 6.2 cm
Weight		
Millenium	175 g	175 g
PowerPack-II	980 g	1 Kg
ProPak-II	1.3 Kg	1.2 Kg
Price		
Millenium	$ 9,000	$ 5,000
PowerPack-II	$ 10,000	$ 6,000
ProPak-II	$ 11,000	$ 7,000
*CEP – Circular Error Probable (circle where 95% of position errors fall into)		
*ppm – Parts Per Million		
*RMS – Root Mean Square		

Table 2.4: Ashtech GPS Receivers.

	Z-12	Z-Xtreme	Z-Eurocard	GG24 Sensor
Position Accuracy				
real-time differential	< 1 m	< 1 m	3.0 m CEP	3.2 m CEP
real-time kinematic	3 cm + 2 ppm H 5 cm + 2 ppm V	1 cm + 2 ppm H 2 cm + 2 ppm V	1.0 cm (1 Hz) 2.0 cm (10 Hz)	35.0 cm CEP
post-processed	1 cm + 1 ppm H 2 cm + 1 ppm V	1 cm + 1 ppm H 2 cm + 1 ppm V	NA	NA
Channel Tracking	12 ch L1/L2	12 ch L1/L2	36 ch L1/L2	36 ch L1/L2
Position Update Rate	1 Hz to 10 Hz	1 Hz to 10 Hz	1 Hz to 10 Hz	1 Hz to 2 Hz
RTK Range		< 10 km	NA	NA
Interface	RS-232	RS-232	RS-232	RS-232
Power Consumption	12 W	6 W	7.5 W	3.2 W
Physical Dimensions (cm)	21.6 x 9.9 x 20.3	7.6 x 20 x 22.5	1.5 x 10 x 17	6 x 17 x 23
Weight	3.85 kg	1.7 kg	0.22 kg	1.6 kg
Price	$ 20K - 26 K	$ 14K - 15.5K	$ 9K - 12K	$ 7K - 9K

Table 2.5: Trimble GPS Receivers.

	Military GPS Survey	4600 LS	4700
Position Accuracy			
standalone	16 m CEP	NA	NA
real-time differential	1 m horiz, 1 m vert	0.2 m + 1 ppm	0.2 m + 1 ppm
real-time kinematic	1 cm + 2 ppm horiz, 2 cm + 2 ppm vert	1 cm + 1 ppm horiz, 2 cm + 1 ppm vert	1 cm + 1 ppm horiz, 2 cm + 1 ppm vert
post-processed	1 cm + 2 ppm horiz 2 cm + 2 ppm vert	1 cm + 1 ppm horiz, 2 cm + 1 ppm vert	1 cm + 1 ppm horiz, 2 cm + 1 ppm vert
Channel Tracking	L1 / L2 + carrier	12 L1 + carrier	12 L1 / 12 L2 + carrier
Power Consumption	9 W	< 1 W	6 W
Physical Dimensions	24.8 x 28 x 10.2 cm	22.1 Dia. x 11.8 cm	11.9 x 6.6 x 20.8 cm
Weight	3.1 kg	1.4 kg	1.2 kg
Price	NA	NA	NA

Table 2.6: GBX Differential GPS / Beacon Receivers.

	GBX Series	GBX-PRO
Position Accuracy	2-5 m (95 %)	< 1 m (95%)
Channel Tracking	8 or 12 ch L1	12 L1 + carrier
Position Update Rate	NA	10 Hz, 20 Hz
Beacon Channels	2 independent	2 independent
Beacon Frequency Range	283.5 to 325.0 kHz	283.5 to 325.0 kHz
Interface	RS-232C	RS-232C
Power Consumption	4.4 W	4.8 W
Physical Dimensions	15 x 12.5 x 5.1 cm	16.3 x 112.5 x 5.1 cm
Weight	0.73 kg	0.8 kg
Price	$ 1400	$ 3000

Integrated Inertial Navigation System / Global Positioning System (INS/GPS)

An autonomous passive system, such as an inertial guidance system, provides reasonably accurate instantaneous position, velocity, attitude and time (PVAT) output with no dependence on any external man-made device or signal. Therefore, it is neither jammable nor capable of being sensed from outside the vehicle. On the other hand, stand-alone navigation systems, such as GPS, require externally provided electromagnetic signals from ground-based radionavigation aids (NAVAIDS) or from space-based satellites (SV).

Integrated navigation systems combine the best features of both autonomous and stand-alone systems and are not only capable of good short-term performance in the autonomous or stand-alone mode of operation, but also provide exceptional performance over extended periods of time when in aided mode. Integration thus brings increased performance, improved reliability and system integrity, at the expense of increased complexity and cost. Moreover,

outputs of an integrated navigation system are usually digital, thus they are capable of being used by other resources or of being transmitted without loss or distortion [Bie99].

The merging of INS and GPS is mutually beneficial and allows for a more reliable and more accurate positioning system. As described earlier, the inherent drift of the INS necessitates the GPS to provide it with accurate position updates, keeping the "navigation state" current and thereby effectively eliminating the drift. External measurements are obtained and processed through an extended Kalman filter to provide the most probable correction to the navigation state estimate.

In order to obtain the highest accuracy, DGPS is used. However, a lot of INS/GPS systems employ stand-alone GPS for aiding. Such systems are common in aircraft navigation. The use of DGPS introduces the factor of correction transmission integrity, where the radio modem link must be maintained at all times.

Aside from commercially-available INS/GPS systems, there has been a great deal of development of custom INS/GPS systems by universities and research laboratories. The selection of the components are crucial as ease of integration becomes a key factor.

Novatel has recently entered the INS/GPS market with its Black Diamond System (BDS). The BDS fuses a Honeywell HG1700AG11 IMU and a Novatel RT-2 DGPS through a 15-state Kalman filter. This system is capable of outputting high accuracy position and orientation data at rates up to 100 Hz. The BDS is discussed in more detail in Chapter 6.

Types of Vehicle Applications

The characteristics and limitations of each positioning system type greatly determines its application in autonomous vehicle navigation

Indoor Vehicles

Indoor vehicles typically use an encoder-compass positioning system set-up or an active beacon type system using ultrasonics or light sources. The amount of slippage between the vehicle's wheels and the surface it travels on is minimal, making encoders ideal for measuring position. A compass is used to account for changes in orientation

An example of an autonomous ground vehicle (AGV) that uses an encoder-compass positioning system is the Cybermotion K2A robot (see Figure 2.5) which was automated by CIMAR.

Figure 2.5: UF's Cybermotion K2A.

The University of Oxford has developed a navigation for a low speed, indoor AGV suitable suitable for industrial use. This system uses SONAR to detect obstacles, comparing it

to a map supplied by the user containing all the possible obstacles to determine current vehicle position. Odometer information is also used to calculate vehicle position. An external Kalman filter is then used to integrate these two estimates of the vehicle position [Ste95].

The University of Michigan has developed a benchmark test for odometric accuracy of mobile robots, called UMBmark. They tested six different vehicle configurations: (1) TRC Labmate, differential drive; (2) Cybermotion K2A synchro drive; (3) CLAPPER MDOF; (4) Remotec Andros; (5) Remotec Andros with encoder trailer; and (6) Smart Encoder Trailer Simulation.

Outdoor Vehicles

Outdoor vehicles are subject to a wider range of operating conditions. As such is the case, the vehicle positioning system must be able to adapt well to changing environment such as terrain variations and weather effects. The terrain variations make wheel encoders less effective due to slippage and non-consistent ground contact forces. The use of Inertial Measurement Units is ideal since they are self-contained and are hardly affected by external disturbances. Also, given good satellite signal reception, Global Positioning Systems play a major role in the positioning of outdoor AGVs. Furthermore, RF based beacon type systems are less subject to interference and their long range make them suited for use on outdoor vehicles.

Carnegie Mellon University, as part of their CyberScout project, has developed two Unmanned Ground Vehicles (UGV) called Lewis and Clark. These are retrofitted Polaris Sportsman 500 All-Terrain Vehicles (see Figure 2.6). Navigational sensing is performed by a 20-cm resolution NovAtel RT-20 DGPS unit. Currently each vehicle is equipped with five cameras, a panning stereo pair in front for obstacle avoidance and mapping, and three pan/tilt

cameras for surveillance, one each located at the front left, front right, and rear. Stereo vision is employed to have a 3-dimensional understanding of the environment to perform free space mapping, thus enabling obstacle detection and path-planning [Dol].

Figure 2.6: Carnegie Mellon University's Cyber ATV.

Researchers at the University of Michigan are using two autonomous vehicle test platforms. The original vehicle, called MAVERIC, is an electric golf cart that contains a 486 PC to perform low level vehicle control, a DataCube hooked up to a CCD camera to grab images, wheel encoders to calculate distance traveled, and a SUN IPX which runs a reactive planner and navigation behaviors. The MAVERIC robot system architecture consists of four layers: vehicle control layer, behavior control layer, manager layer, and the planner layer [Ken].

The planning layer consists of an implementation of the Procedural Reasoning System planner (PRS) developed at SRI. PRS allows the robot to pursue long-term goals by adopting pieces of relevant procedures based on context rather than blindly following a prearranged plan. The current vehicle testbed is a military all-terrain vehicle, the HMMVW [Ken].

Figure 2.7: University of Michigan's MAVERIC.

Figure 2.8: University of Michigan's HMMVW.

An excellent case where GPS is relied on solely for vehicle positioning is in the development of an autonomous John Deere 780 agricultural tractor by Stanford University. The position and orientation are computed using a four single-phase GPS antenna array mounted on

orthogonal axes defining the vehicle roll and pitch. High accuracy position and orientation are obtained by using differential GPS in conjunction with an additional ground station, called a pseudolite, which sends out GPS-like signals. Experimental results have shown position standard deviation to be 2.5 cm while orientation standard deviation to be less than 1 degree [Oco].

Figure 2.9: Stanford University's Robotic John Deere 7800.

Military Applications

The military has a long history of developing GPS and IMU guidance technology for aircraft and weaponry. The first weapon GPS receiver was developed in the late 1970s. More recently, the Tactical GPS Antijam Technology (TGAT) program, completed in 1992, developed an adaptive GSP filter/antenna that simultaneously accomplished jammer discrimination in both the spacial and temporal domains. More recently, a tightly coupled weapon grade IMU/GPS system has been used to develop a low cost high antijam system.

The United States entered into a cooperative Engineering and Manufacturing Development (EMD) program with the United Kingdom, Germany, France and Italy to develop a new guided rocket for the Multiple Launch Rocket System, the M30 GMLRS. The M30 uses a Honeywell IMU and a GPS receiver (10 m. CEP) for navigation and guidance. On February 11, 1999 with a GPS-aided flight test, the M30 flew 49 km and impacted an impressive 2.1 meters from the target center.

CHAPTER 3
MAPS / ASHTECH GPS POSITIONING SYSTEM

On the Navigation Test Vehicle (NTV), the current positioning system used is an integrated INS/GPS system. The Inertial Navigation System is a Honeywell H-726 Modular Azimuth Position System (MAPS) and the Global Positioning System is an Ashtech Z-12 Differential GPS.

The following chapter is based on previous research conducted since 1992 by graduate students working on CIMAR's Autonomous Vehicle Project. This chapter largely contains excerpts from research papers written on the development and implementation of the MAPS/Ashtech system. Furthermore, the chapter contains information from Honeywell's Modular Azimuth Position System technical description manual and the Ashtech Z-12 GPS Receiver manual.

It must be noted that the MAPS/Ashtech positioning system has been extensively utilized and tested since 1996. Further, it has been adapted to several different ground vehicle platforms, which have demonstrated various practical military applications.

Overall System Description

The MAPS/Ashtech positioning system is essentially a GPS-aided INS that is integrated through an external Kalman Filter (KF). The MAPS offers position and orientation data at all times at data rates of 10 Hz. The GPS prevents the MAPS position output from drifting by

constantly updating it with high accuracy DGPS position at lower data rates of 1 Hz. The job of the KF is to predict or estimate a future position and vehicle trajectory based from the INS/GPS model, current and previous position and orientation data. Such estimates increase position accuracy not only during normal operation but more importantly in the event of GPS loss.

Honeywell H-726 Modular Azimuth Position System

The Honeywell H-726 Modular Azimuth Position System is a completely self-contained, strapdown inertial navigation system. The MAPS consists of a Dynamic Reference Unit (DRU), a control display unit (CDU) and a vehicle motion sensor (VMS). The DRU is the heart of the MAPS system providing all necessary navigation data. Given only initial position, the DRU uses three ring laser gyros and three accelerometers combined with support electronics to accurately determine position, angular orientation, velocities and angular rates in real time [Arm93].

System Specifications and Features

The physical dimensions, weight, operational conditions, power requirements, and other features are listed in Table 3.1. Table 3.2 shows the performance and specifications for the MAPS.

System Components

The DRU is shown in Figure 3.1 with its top cover removed. The DRU's basic components consists of an inertial sensor assembly, main processors and support electronics including power supplies.

Table 3.1: Honeywell H-726 Features.

Feature	Honeywell H-726 MAPS
Dimensions (L x W x H)	15” x 10.75” x 8.7”
Weight (lb)	46.7
Housing	Cast Aluminum
Allowable Temperature Range	-46°C to +60°C
Allowable Vibration	Tracked Vehicles
Allowable Shock	Howitzer gunfire
Allowable Angular Rate	400 degrees/second
Voltage Requirements	24VDC (18.5-36.0 VDC)
Power Requirements	103 Watts
Communications Interface	Bi-directional RS422 SDLC

Table 3.2: MAPS Performance versus Specifications.

DRU Performance vs. Specification Requirement

Parameter	MIL-70789 A Spec.	Demonstrated	Conditions
Normal Align	5 min.	<0.35 mil PE Secant Lat	Less than ± 80 deg Lat
Survey Azimuth	12 min.	<0.25 mil PE Secant Lat	Less than ± 80 deg Lat
		<0.33 mil PE Secant Lat	
Stored Align	0.1 mil PE	< 0.1 mil PE	Relative to Stored Heading
Pitch/Roll (Cant)	0.34 mil PE	<0.3 mil PE	
Horizontal Position (Odometer Aided Mode)	10m CEP .0025 Dist. Trav.	3.3 CEP .0012 Dist. Trav.	Distance < 4 km odo mode Distance > 4 km odo mode
Altitude (Odometer Aided Mode)	6.7 CEP .00067 Dist. Trav.	3.0 CEP .00052 Dist. Trav.	Distance < 10 km odo mode Distance > 10 km odo mode
Horizontal Position (ZUPT Mode, No Odometer Aiding)	18m CEP	9.0m CEP	Distance ≤ 27 km
Altitude (ZUPT Mode, No Odometer Aiding)	10m CEP	2.6m CEP	Distance ≤ 35 km

* Note : ZUPT data at four minute ZUPT intervals. Data is essentially equal at ten minute intervals.

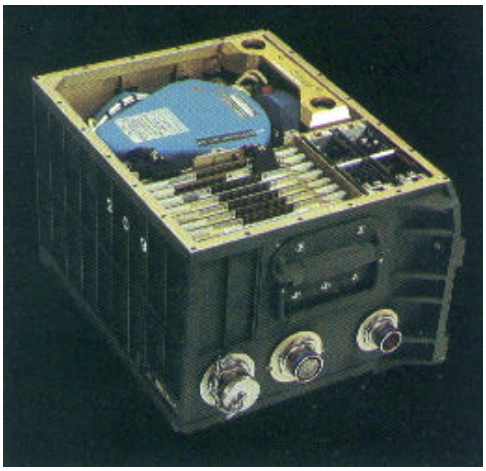


Figure 3.1: Honeywell H-726 MAPS Dynamic Reference Unit (DRU).

The inertial sensor assembly (ISA) is shown in Figures 3.2 and 3.3. It is a removable calibrated assembly containing the inertial components necessary for measuring the rotation angles and accelerations in three-dimensional space. It includes the following subassemblies:

1. Three GG1342 Ring Laser Gyros (RLGs)
2. Three Sundstrand QA2000 Q-FLEX accelerometers
3. Temperature/calibration coefficient PROM
4. High voltage power supply
5. ISA precision sensor block

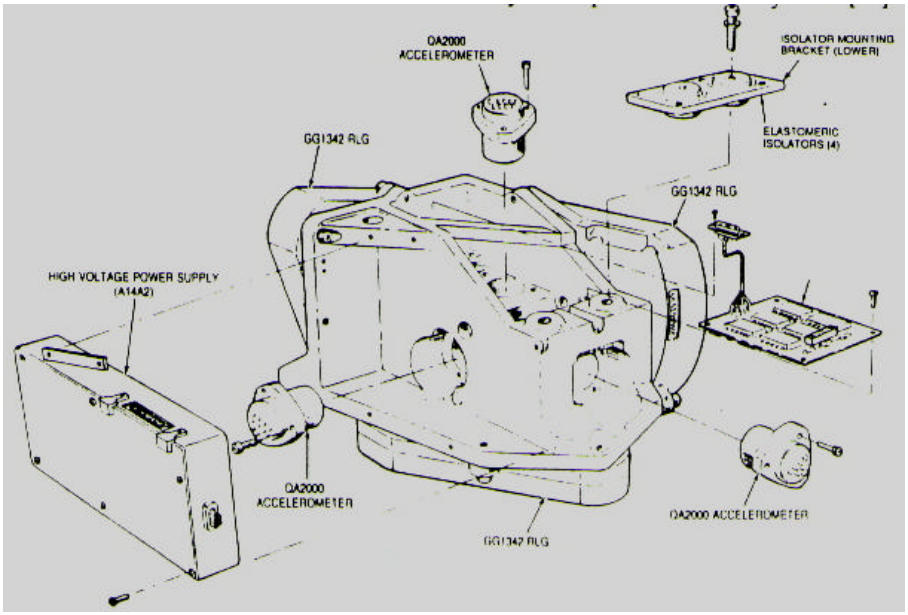


Figure 3.2: MAPS DRU Inertial Sensor Assembly Schematic.

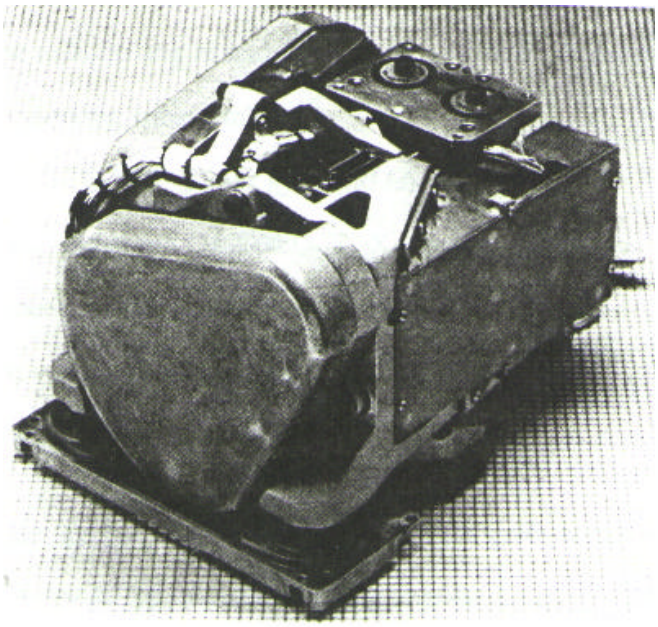


Figure 3.3: MAPS DRU Inertial Sensor Assembly.

Ring Laser Gyros. Each of the three RLGs contained in the DRU is a single axis device capable of measuring angular rotations and angular rates about the input axis. The three gyros are

mounted mutually perpendicular and co-linear with the accelerometers on the coordinate axes allowing measurements in three-dimensional space.

The RLG does not use a spinning mass as used in most conventional gyros. The RLG is a Sagnac interferometer. The device is a laser that incorporates three reflectors, arranged so that the light beams form an enclosed triangle. The reflecting mirrors, together with the light-amplifying medium in the light path, constitute the oscillator. Two such oscillators propagate light energy along the same path but in opposite directions. The frequencies at which these oscillators operate are determined by the length of the optical path. If the ring laser is stationary in inertial space, the clockwise (cw) and counterclockwise (ccw) beams oscillate at the same frequency. Rotation causes a slight difference in the cw and ccw path lengths, and consequently the wavelengths and oscillating frequencies. These differences in frequency are proportional to inertial angular rate. Figure 3.4 shows a schematic of the RLG [Tit97].

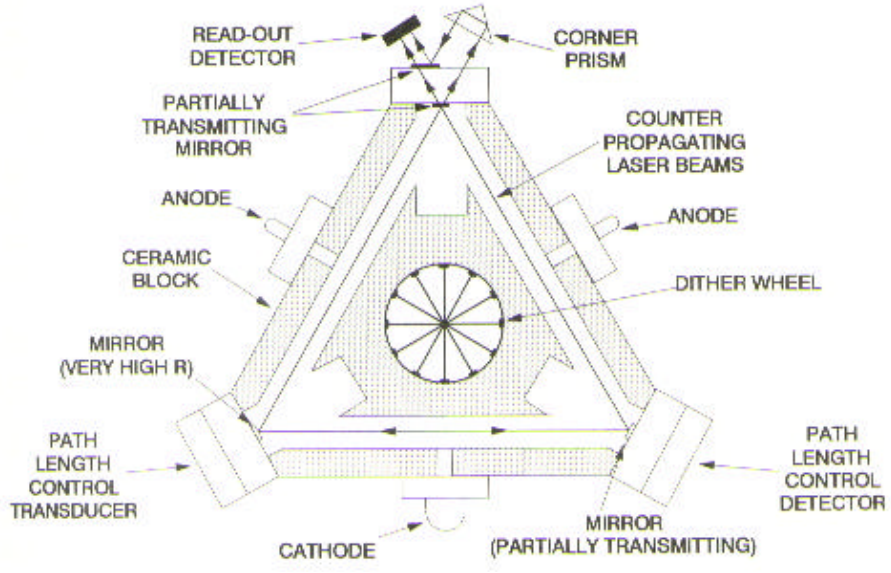


Figure 3.4: Ring Laser Gyroscope Schematic.

Accelerometers. Three single-axis accelerometers are mounted mutually perpendicular as with the three RLGs. These sensors measure the acceleration on each axis which can then be integrated to determine velocities and displacements.

Figure 3.5 shows a schematic of the QA2000 accelerometer. Acceleration is determined using a common force balance technique as follows: A quarts-flexure suspension supports a torquer coil/seismic element that functions as the moving element of a differential capacitor pick-off. An input acceleration causes deflection of the seismic element and capacitor plate, producing an AC output voltage. This voltage is fed back to the torquer coil producing an electromagnetic torque to balance the inertial effects of the sensed acceleration. The feedback current required to maintain the pendulum at a balanced position is directly proportional to the acceleration.

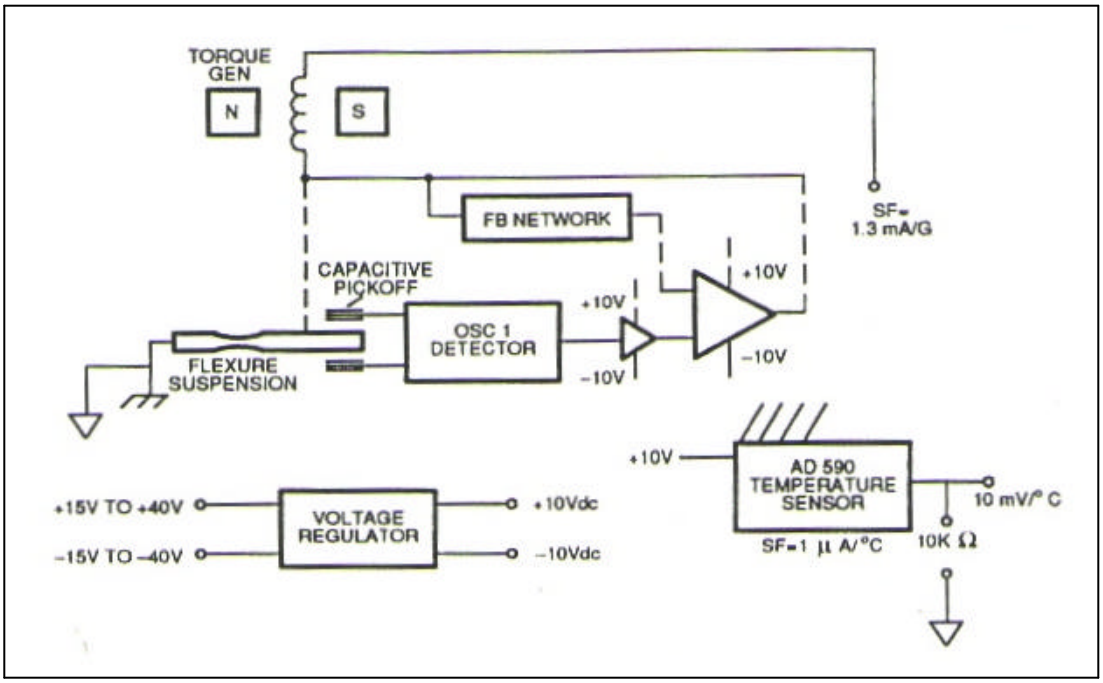


Figure 3.5: QA2000 Accelerometer Schematic.

The software tasks performed by the DRU are partitioned onto two STD-1750A processors, the inertial processor (IP), and the survey processor (SP). Figure 3.6 shows a breakdown of the tasks performed by each processor.

The primary function of the IP is to process raw inertial data that is outputted directly from the gyros and accelerometers. The IP uses the 1750A assembly language, that is common to all Honeywell RLG products, to handle the high through-put and computational rates required.

The survey processor takes compensated velocity and rate data from the IP and computers survey and positioning parameters. In addition, it performs built-in tests (BITs), monitors other system functions, and processes commands received over the two data buses. Because the speed demands are less crucial, the SP uses the JOVIAL J73B high-order programming language [Arm93].

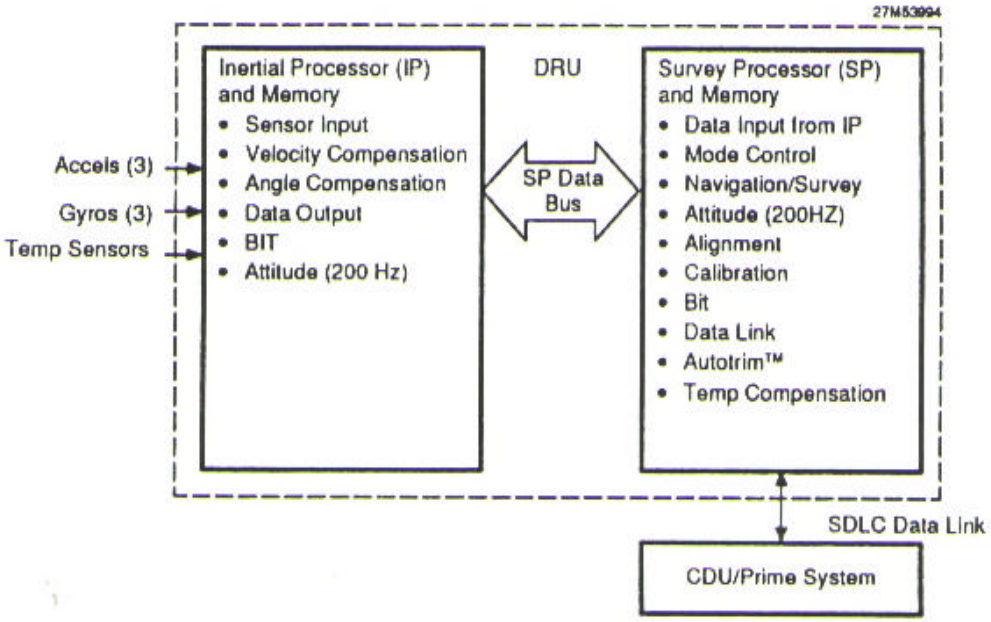


Figure 3.6: Inertial Processor (IP) and Survey Processor (SP) Commands.

The DRU's support electronics include gyro and accelerometer electronic assemblies, A/D-Pulse accumulator module, I/O interface, thermal sensors, and power supplies. The DRU's functional block diagram is depicted in Figure 3.7.

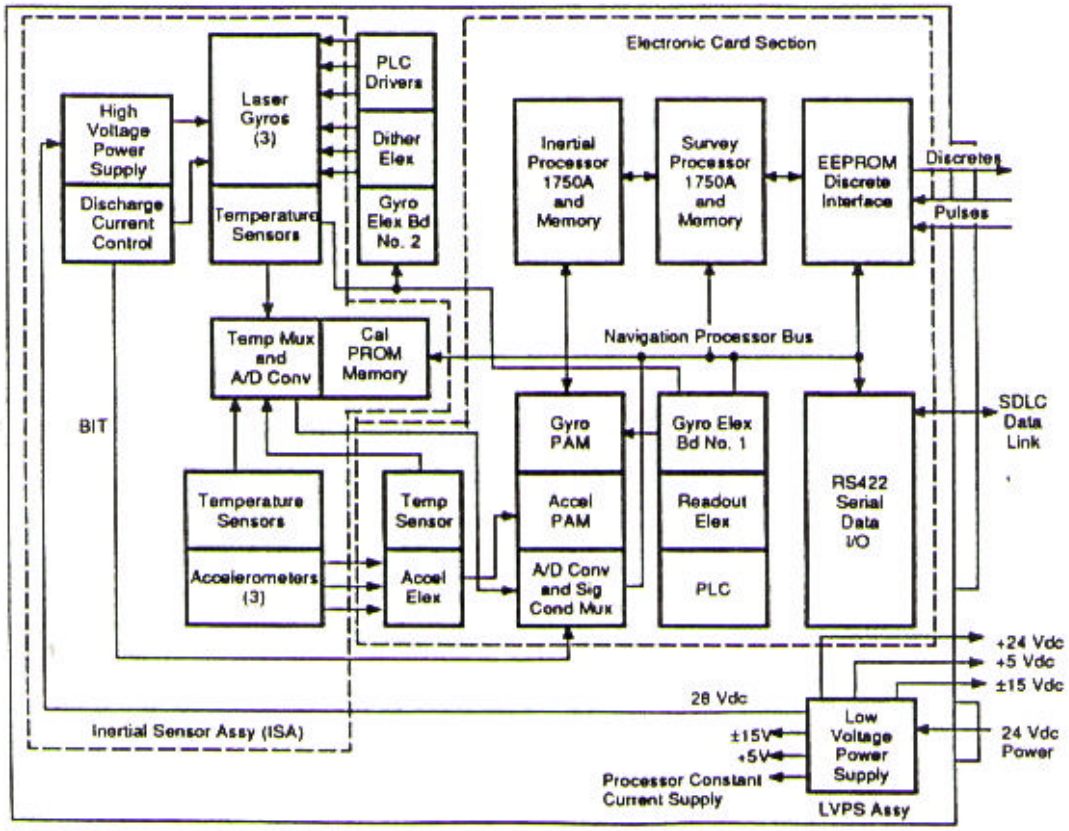


Figure 3.7: DRU Functional Block Diagram.

Commands and Messages

Table 3.3 list the DRU's commands, command rates and response times. Table 3.4 defines the data messages.

Table 3.3: DRU Commands.

Command	Maximum Request Rate	Response Message	Maximum Response Time	DRU Spec Para Ref – Commands
A. Commands for Data Transfer Prime System or CDU to DRU				
(1) ACCEPT POSITION (NAVIGATION DATA)	5/sec	STATUS DATA	130 msec	10.1.1
(2) ACCEPT HEADING (GUN GRID AZIMUTH)	5/sec	STATUS DATA	130 msec	10.1.2
(3) ACCEPT INSTALLATION/VEHICLE DATA	5/sec	STATUS DATA	130 msec	10.1.3
B. Commands for Data Transfer DRU to Prime System or CDU				
(1) RETURN DRU RATE DATA	100/sec	DRU RATE DATA	10 msec	20.1.1
(2) RETURN CONFIGURATION DATA	5/sec	CONFIGURATION DATA	130 msec	20.1.2
(3) RETURN STATUS	5/sec	STATUS DATA	130 msec	20.1.3
(4) RETURN NAVIGATION DATA	5/sec	NAVIGATION DATA	130 msec	20.1.4
(5) RETURN ATTITUDE DATA	100/sec	ATTITUDE DATA	10 msec	20.1.5
(6) RETURN ALIGN TIME TO GO	5/sec	ALIGN TIME TO GO DATA	130 msec	20.1.6
(7) RETURN ALERT DATA	5/sec	ALERT DATA	130 msec	20.1.7
(8) RETURN BIT DATA	5/sec	BIT DATA	130 msec	20.1.8
(9) RETURN TRAVEL LOCK DATA	100/sec	TRAVEL LOCK DATA	10 msec	20.1.9
(10) RETURN POSITION DATA	5/sec	POSITION DATA	130 msec	20.1.10
(11) RETURN DRU ORIENTATION DATA AND POINTING DEVICE DATA	100/sec	DRU & POINTING DEVICE ORIENTATION DATA	10 msec	20.1.11
(12) CONTRACTOR RESERVED COMMANDS	5/sec	VARIOUS	130 msec	20.1.12
(13) RETURN BORESIGHT	5/sec	BORESIGHT ANGLES	130 msec	20.1.13
(14) RETURN EXPANDED STATUS	5/sec	EXPANDED STATUS DATA	130 msec	20.1.14
(15) RETURN SURVEY QUALITY	5/sec	STATUS DATA	130 msec	20.1.15
C. Commands to Control DRU Mode or Operation				
(1) OVERRIDE ALERT	5/sec	STATUS DATA	130 msec	30.1.1
(2) RESET DISTANCE	5/sec	STATUS DATA	130 msec	30.1.2
(3) RESTART	5/sec	STATUS DATA	130 msec	30.1.3
(4) SPARE	N/A	N/A	N/A	N/A
(5) SPARE	N/A	N/A	N/A	N/A
(6) INHIBIT AUXILIARY BUS CONTROL	5/sec	STATUS DATA	130 msec	30.1.6
(7) ENABLE AUXILIARY BUS CONTROL	5/sec	STATUS DATA	130 msec	30.1.7
(8) STORED HEADING	5/sec	STATUS DATA	130 msec	30.1.8
(9) REALIGN	5/sec	STATUS DATA	130 msec	30.1.9
(10) SHUTDOWN	5/sec	STATUS DATA	130 msec	30.1.10
(11) OUT OF TRAVEL LOCK	5/sec	STATUS DATA	130 msec	30.1.11
(12) INHIBIT ZERO-VELOCITY UPDATE	5/sec	STATUS DATA	130 msec	30.1.12
(13) ENABLE ZERO-VELOCITY UPDATE	5/sec	STATUS DATA	130 msec	30.1.13
(14) IN TRAVEL LOCK	5/sec	STATUS DATA	130 msec	30.1.14
(15) ZUPT MODE REQUEST	5/sec	STATUS DATA	130 msec	30.1.15
(16) ODOMETER MODE REQUEST	5/sec	STATUS DATA	130 msec	30.1.16
(17) FIX ALTITUDE	5/sec	STATUS DATA	130 msec	30.1.17
(18) RELEASE ALTITUDE	5/sec	STATUS DATA	130 msec	30.1.18

Table 3.4: DRU Data Messages.

Data Message	DRU Spec Para Ref
(1) DRU RATE DATA	40.1.1
DRU Azimuth Rate	
DRU Pitch Rate	
DRU Roll Rate	
(2) Configuration Data	40.1.2
α, β, γ, A, B, Γ	
ΔX, ΔY, ΔZ, ZUPT Intervals, Align Times, Realign Time	
Configuration Code, Configuration Flags (8)	
VMS Scale Factor, Fuel Consumption Factor	
DRU Coordinate Frame Rotation Matrix [R]	
(3) NAVIGATION DATA	40.1.3
Northing	
Altitude	
Distance Traveled	
Pointing Device Geodetic or Grid Azimuth	
Vehicle Cant or Roll	
Vehicle Pitch	
Vehicle Geodetic or Grid Azimuth	
DRU Spheroid	
Easting	
Hemisphere and Zone of DRU	
(4) ATTITUDE DATA	40.1.4
Printing Device Geodetic or Grid Azimuth	
Pointing Device Elevation	
Pointing Device Azimuth Rate	
Pointing Device Elevation Rate	
Vehicle Cant or Roll	
Vehicle Pitch	
(5) ALIGN TIME TO GO DATA	40.1.5
Align Time To Go	
(6) ALERT DATA	40.1.6
(7) BUILT–IN–TEST (BIT) DATA	40.1.7

Data Message	DRU Spec Para Ref
(8) TRAVEL LOCK DATA	40.1.8
Pointing Device Geodetic or Grid Azimuth	
Pointing Device Elevation	
Travel Lock Geodetic or Grid Azimuth Reference	
Travel Lock Elevation Reference	
Pointing Device Azimuth Rate	
Pointing Device Elevation Rate	
(9) POSITION DATA	40.1.9
Spheroid	
Hemisphere and Zone	
Easting	
Northing	
Altitude	
(10) DRU ORIENTATION & POINTING DEVICE DATA	40.1.10
DRU Geodetic or Grid Azimuth	
DRU Pitch	
DRU Roll	
Pointing Device Geodetic or Grid Azimuth	
Pointing Device Pitch	
Pointing Device Cant	
(11) STATUS DATA	40.1.11
(12) BORESIGHT ANGLES	40.1.12
A	
B	
Γ	
(13) EXPANDED STATUS DATA	40.1.13
DRU Operating In Extended UTM Zone	
Altitude Clamped	
(14) SURVEY QUALITY	40.1.14
Filter Estimate of Azimuth Error (1σ)	
Filter Estimate of Position Error (1σ)	
Filter Estimate of Altitude Error (1σ)	

Modes of Operation

The MAPS DRU has four basic modes of operation:

Power-Up : The DRU performs a series of built-in tests (BIT) and initialization procedures to ensure that all systems are functioning properly.

Align : The DRU establishes a directional reference by self-leveling and performing a gyro-compassing alignment to true north. A 'Normal Align' mode, which typically takes 10-15 minutes, uses external position inputs to calculate its orientation. A 'Stored Heading Align' mode uses previously stored values of position and attitude from last power down to speed up the alignment process.

Survey : After the first 3 minutes of a normal align or upon completion of a stored heading align, the DRU continuously updates position, orientation and angular rates. Velocity errors, which grow with time, are countered by 'Exclusive ZUPT mode' and 'Odometer mode'. In the Exclusive ZUPT mode, the DRU, which has to be in a stationary or rest position, resets the velocity errors to zero. In the Odometer mode, the DRU uses both the ZUPT information and external odometer pulse information to damp velocity errors.

Power-Down : As soon as the host computer sends a shutdown command, the DRU will store all pertinent information and orientation data and perform BITs before shutting down system power.

Ashtech Z-12 GPS Receiver

The Global Positioning System component selected for integration was the Ashtech Z-12 GPS receiver (Figure 3.8). This functionality, performance and rugged design of this receiver is ideal for military applications. The differential set-up is comprised of a Z-12 receiver as a remote receiver (on the NTV) and a second Z-12 receiver as a base receiver.

Figure 3.8: Ashtech Z-12 GPS Receiver.

System Specifications

It has twelve independent channels an can track all of the satellites in view automatically. It uses carrier-phase differential to estimate remote receiver position to within 6 centimeter accuracy. Figure 3.9 shows a dynamic test of the Ashtech Z-12 in carrier phase differential mode. The output position data rate is 1 Hz.

System Features

Aside from Real-Time Kinematic position solution updates (while in carrier-phase differential mode), the Ashtech Z-12 offers numerous useful features. A front-panel display allows the user to view information from position data, satellite-in-view information and availability, and even allows storage of survey data in multiple files. These data files can then be downloaded and post-processed using proprietary Ashtech software, PNAV, to calculate position to within 2 millimeter accuracy. The current receiver software has been recently

upgraded to eliminate computational errors brought about by the turnover from year 1999 to 2000 (Y2K). Each Ashtech Z-12 receiver also has its own GPS antenna with choke ring, a radio modem for transmitting / receiving differential corrections and a 12-volt power supply.

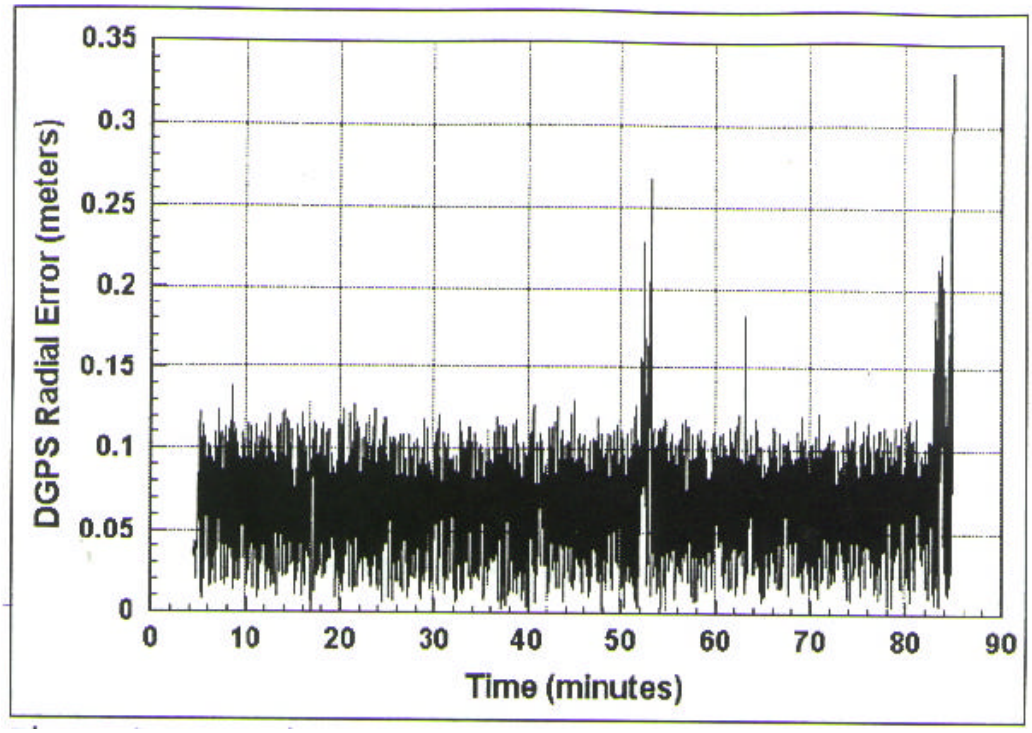


Figure 3.9: Dynamic Test of Ashtech Z-12 DGPS Receiver.

Message Sets

The essential position data in geodetic coordinates is logged out of the remote receiver at 1 Hz. This real-time data log is output as a "Cben" type message in ASCII format. The format of the position message fields is given below:

→ "$PASHR,CBN," header string

→ Receive time: GPS time in seconds of week when code was received

→ Station position: Latitude (degrees)

→ Station position: Longitude (degrees)

→ Station position: Altitude (meters)

→ Velocity in East direction (m/sec)

→ Velocity in North direction (m/sec)

→ Velocity in Up direction (m/sec)

→ RMS position error (meters)

→ Number of Satellites used for position computation

→ PDOP, Position Dilution of Precision

System Integration

Integration of the MAPS and the Ashtech Z-12 GPS receiver was done by using CIMAR's MAX Architecture standards. The MAPS/Ashtech system is a fully-independent and self-contained unit. The components are put on rack-mounted shelves. The primary shelf includes a Z-12 GPS remote receiver, a Freewave radio modem, and the primary host computer (POS PC-104 stack). The secondary shelf houses the H-726 MAPS and a 12V-24V DC-DC power converter. Figure 3.10 shows the present hardware set-up.

System Configuration

The system schematic is depicted in Figure 3.11. Tables 3.5, 3.6 and 3.7 lists the configuration of the MAPS DRU, Z-12 receiver, and Freewave radio respectively.

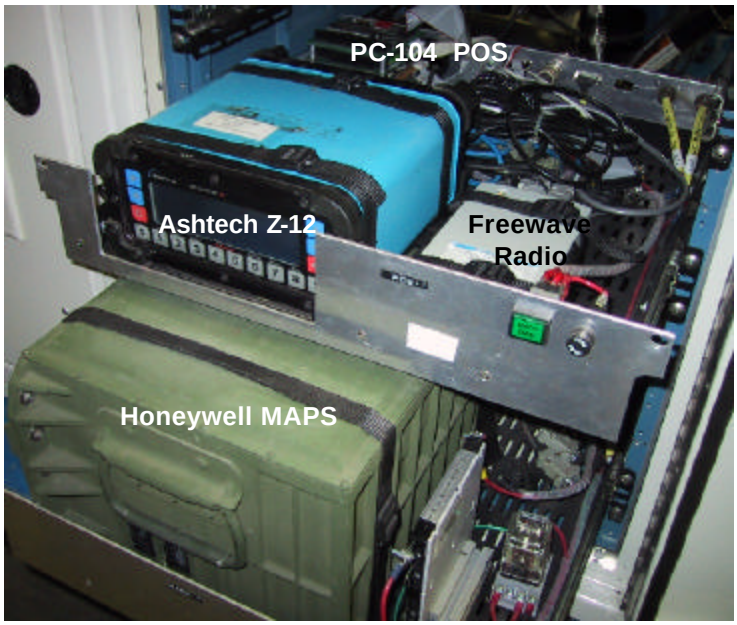


Figure 3.10: MAPS/Ashtech GPS POS Shelf.

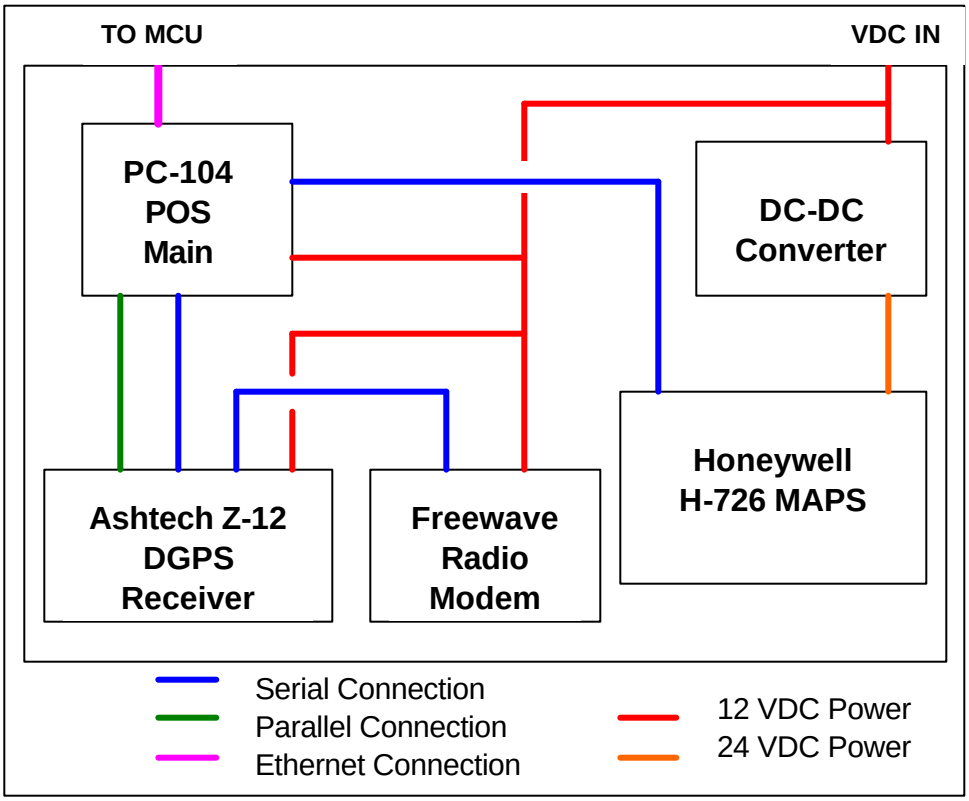


Figure 3.11: MAPS/Ashtech GPS Schematic.

Table 3.5: MAPS DRU Configuration Settings.

Parameter	Current Setting
Velocity Damping Mode	Exclusive ZUPT
ZUPT Interval	4.0 minutes
Normal Align Time	15.0 minutes
Stored Heading Align Time	1.5 minutes
Tracked / Wheeled Vehicle	Wheeled

Table 3.6: Ashtech Z-12 GPS Remote Receiver Settings.

Parameter	Current Setting
Survey Mode	Carrier Phase Differential (RTK)
Position Message Type	cben
Position Message Output Rate	1 Hz
Differential Corrections Rate	1 Hz
Serial Port A Data Rate	19600 bps
Serial Port B Data Rate	19600 bps
1 PPS Output	ON
Minimum Satellites	4

Table 3.7: Freewave Data Transceiver Settings.

Parameter	Current Setting
Data Rate	19600 bps (baud)
Data Bits	8
Parity	None
Stop bits	1
Transmission Frequency	926.5 Khz
Frequency Key	5

Communications Interface

Table 3.8: Communications Configuration.

Parameter	POS to MCU	MAPS to POS	GPS to POS	Radio to GPS
Interface Type	RS-232	RS-422 SDLC	RS-232	RS-232
Data Rate	19600 bps	38400 bps	19600 bps	19600 bps
Data Flow	bi-directional	bi-directional	GPS → POS	Radio → GPS
Port A	POS COM1	MAPS Main	GPS Port A	Radio
Port B	MCU COM3	POS COM3	POS COM2	GPS Port B
Hardware	-	Sealevel ACB104 using polling	-	-

System Control

The primary or host computer (POS) contains the software for integration. The system control is divided into five areas: MAPS, GPS, Filter, Sysman, and Hostcom. All programs are written in C language running on Lynx RT (Real Time) Operating System.

MAPS : This handles all communications between H-726 MAPS and host computer. This includes receiving and processing of MAPS position and orientation data, sending of position updates and system commands.

GPS : This block reads in GPS position messages and stores the data in shared memory. Also, a 1 PPS (Pulse Per Second) signal is used to reset the PC104 system clock synchronizing it with GPS time.

Filter : This part includes the external Kalman filter code and inputs from MAPS and GPS. The output position and orientation data is then sent through Hostcom to the MCU.

Sysman : This takes charge of monitoring all the processes performed by the MAPS, GPS, and Filter programs.

Hostcom : This handles all data transferred to and from the POS and the MCU.

Debug : This program allows the user to run the POS system independently during testing or system debugging phase.

POSConfig.cfg : This file contains the position and orientation offsets (in vehicle coordinates) of each sensor, namely: (1) MAPS to Control Point and (2) GPS to Control Point. These offsets are critical in accurately referencing sensor position thereby allowing more effective vehicle control.

Kalman Filter Design and Implementation

An external filter is used to integrate the MAPS and DGPS position data in order to form a navigation solution. The filter used is a linear discrete Kalman filter. The filter includes nine states for the MAPS' position, position rate and tilt errors. Since the filter is being used for a ground vehicle, it is implemented in a local level geodetic frame. As a result, other MAPS errors can be included as process noise terms in the filter as tuning parameters. The filter processes the DGPS data by alternating between position and change in position [Rog96].

Timing

The DGPS and MAPS data that are being used to calculate the error in the MAPS data must be valid at the same time, reaching an accuracy of timing in the order of one-hundredth of a second [Wit96]. Since only the GPS data includes a time stamp, expressed in seconds of the week in Greenwich Mean Time (GMT), it is used as the reference to which the MAPS data will be tied to. The synchronization is carried out using the GPS receiver's 1 PPS output that sends a TTL signal into a 75 ohm-impedance and signals the exact time the GPS position data is valid. The 1 PPS is then read in by the primary POS PC104 stack through a parallel port, thereby also setting the system clock.

Once the POS clock is synchronized with GPS time, the current time is accessed and stored immediately before the MAPS unit is polled for its position and orientation data. The latencies to be accounted for include time it takes to: (1) transmit message requesting MAPS to send data message, (2) MAPS to acknowledge send command since MAPS is updating at 12.5 Hz, and (3) MAPS to transmit data message to host computer.

To counter this, the sum of these latencies are experimentally determined by matching the DGPS position with MAPS position using turns as reference points. The difference in time between the two systems at that reference point is then used as an offset, effectively synchronizing the whole system.

System Performance

The navigation filter assumes that the position data from the DGPS is accurate to within a tenth of a meter. It uses the DGPS data to build an error model of the MAPS position, position rate, and tilt errors. The accuracy of the filter is highly dependent on the accuracy and availability of the DGPS data. The purpose of the error model is to calculate the position even in the event of loss of DGPS data, and to check the quality of the DGPS data before it processes this data.

The accuracy of the filter is tested using several scenarios that may occur which could compromise system performance. First, the system is tested under ideal conditions. Second, temporary loss of DGPS data will be tested. And lastly, complete loss of DGPS data will be simulated. All tests are conducted at Flavet field at the University of Florida. The GPS base station is within 125 meters of the vehicle at all times. MAPS, DGPS, and filter position data

are recorded during each test. The error in each position data set is calculated by comparing them to the results of post-processed GPS data [Wit96].

Testing Under Ideal Conditions

To test conditions when DGPS data always available and to within required accuracy, the NTV is allowed to survey in an open area free from GPS signal obstructions. In Figure 3.12, the MAPS and DGPS position data are plotted against each other. Here the drift in the MAPS position is quite clear.

Figure 3.13 is a plot of the position data output of the filter. As shown in Figure 3.14, the average error is 7 centimeters.

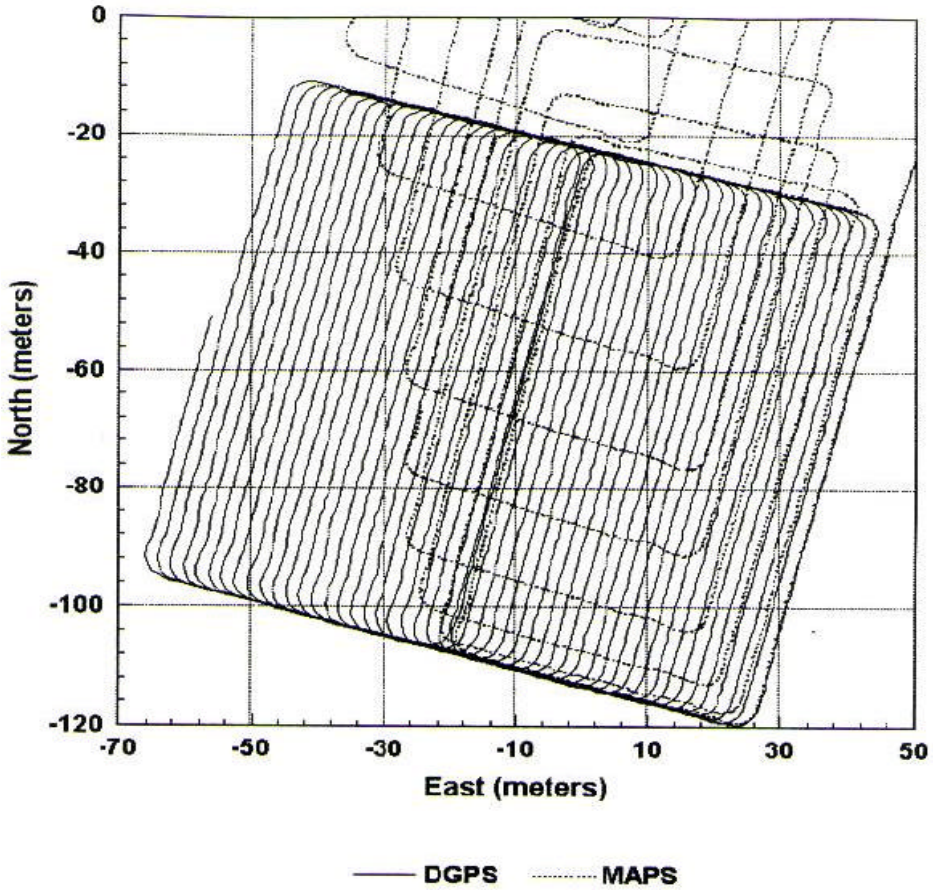


Figure 3.12: DGPS and MAPS North versus East.

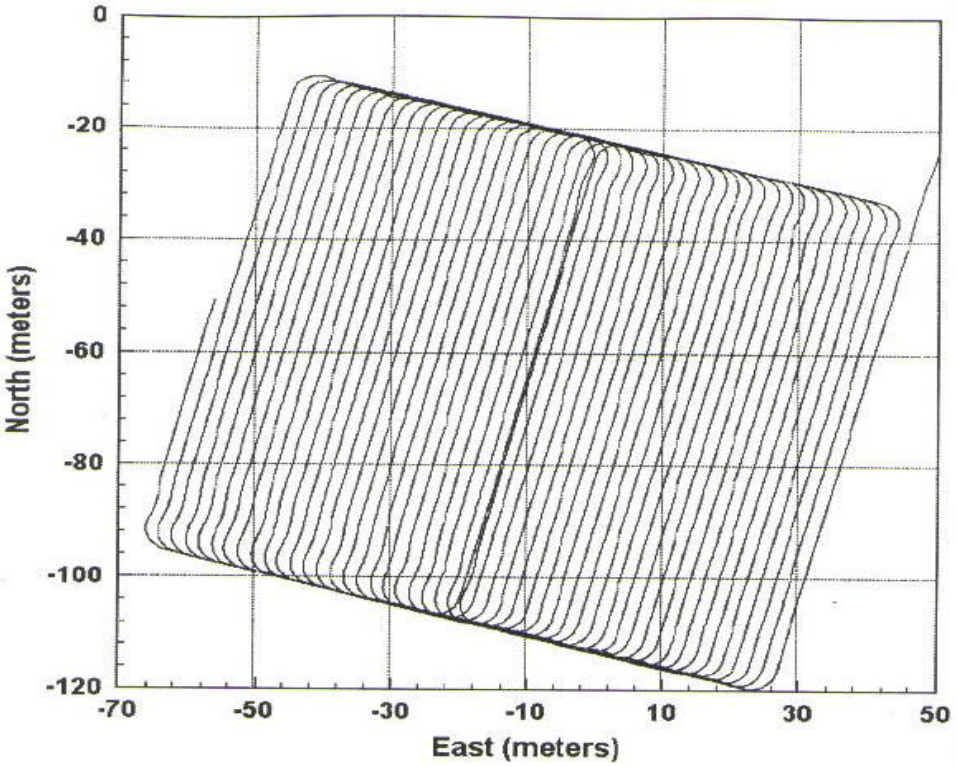


Figure 3.13: Filter North versus East.

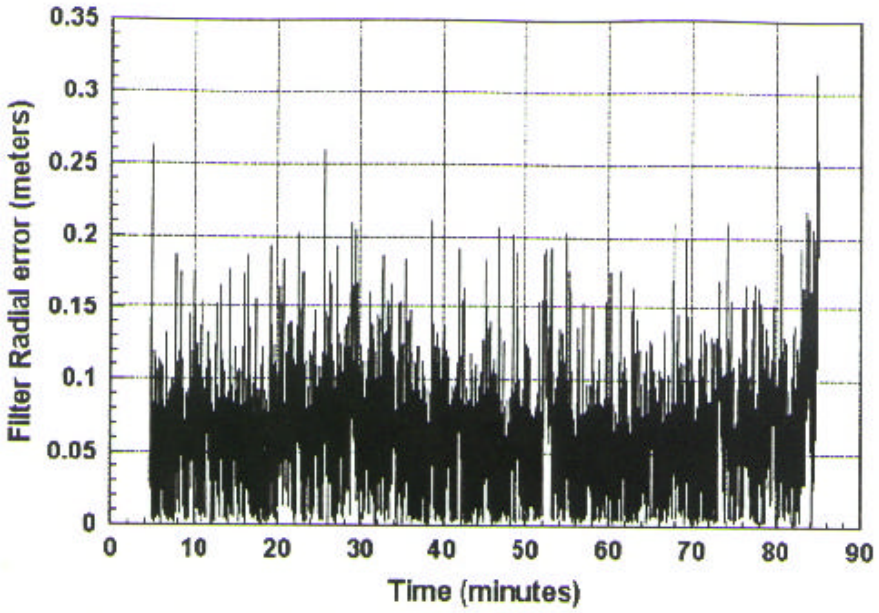


Figure 3.14: Filter Radial Position Error versus Time.

Testing Under Adverse Condition

The presence of redundancy in the system plays a major role when the system is subjected to less than ideal conditions. As pointed out earlier, GPS loss is the biggest concern. Just how the Kalman Filter compensates for GPS outages is tackled in the successive sections.

Temporary Loss of DGPS data

When the NTV travels is in a place which does not permit direct line-of-sight with at least 4 satellites, such as driving under trees or beside tall buildings, there is said to be temporary loss of GPS data.

To simulate loss of GPS data, three specific areas inside the survey field are marked off. When the NTV passes through these marked areas, DGPS data is withheld from the Kalman filter. Figure 3.15 shows the survey field, the marked areas and position data output of the filter. As seen from the results listed in Table 3.9, the temporary loss of GPS data does not significantly reduce the accuracy of the Kalman filter navigation solution.

Table 3.9: Deviation of Real Time Data from Post-Processed DGPS Data

	Temporary GPS Loss	Ideal Conditions
Average Deviation (m)	0.06	0.06
Maximum Deviation (m)	0.32	0.32
Standard Deviation (m)	0.04	0.03

Complete Loss of DGPS data

Complete loss of GPS data occurs when the NTV goes out of the communication range of the base station thereby losing the differential corrections. This will cause the GPS position accuracy to rapidly grow and thus be discarded by the filter.

To test this case, the NTV is first allowed to navigate autonomously for fifteen minutes, during which the filter constructs a model of the MAPS errors. After 15 minutes, the GPS data is withheld from the filter. The NTV is then allowed to navigate for another 5 minutes. In the succeeding two tests, the amount of time used to construct the MAPS error is 30 and 45 minutes respectively.

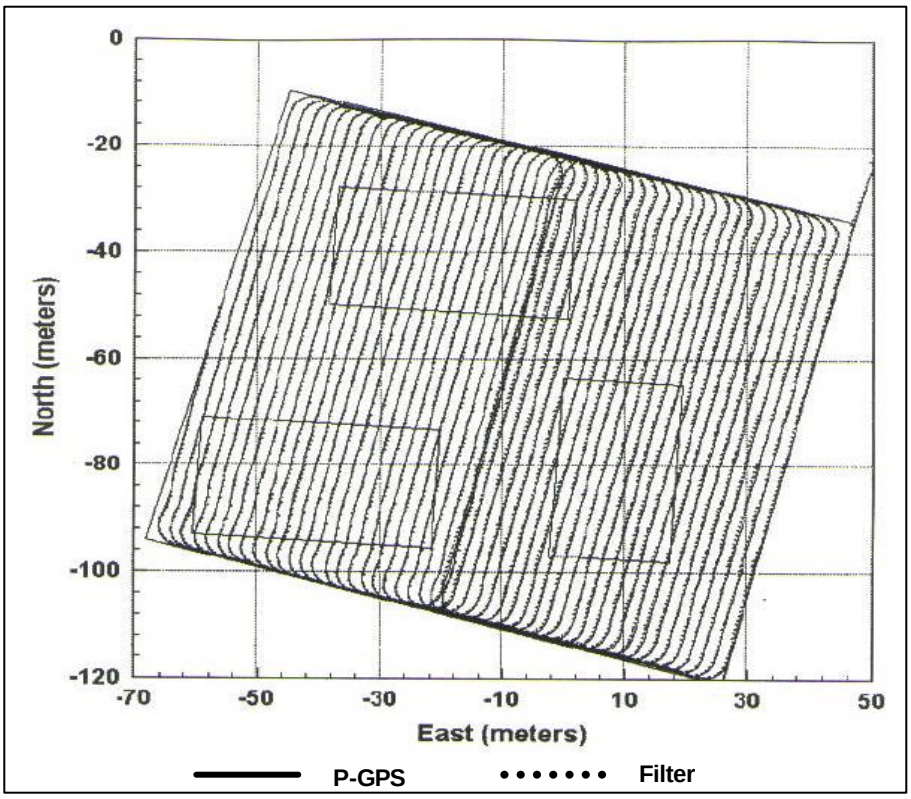


Figure 3.15: P-GPS and Filter North versus East
With Temporary Loss of DGPS Aid.

A number of test runs for each case gave similar results. They are summarized in Figures 3.16 and 3.17. In each of these tests, the filter is able to maintain an accuracy of approximately one meter for two minutes after the loss of GPS data [Wit96].

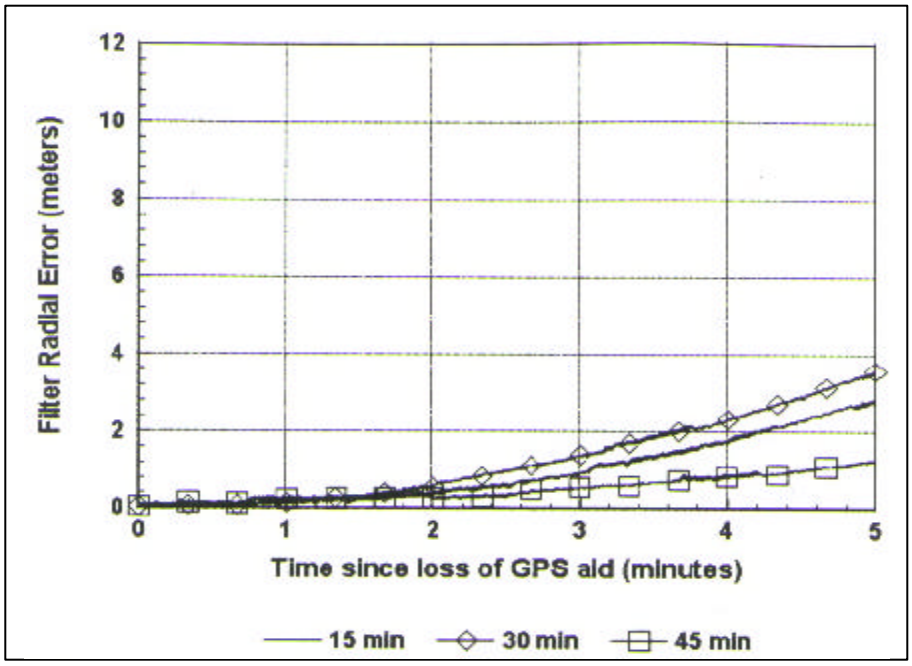


Figure 3.16: Filter Radial Error versus Time Since Loss of DGPS Aid – Best Case.

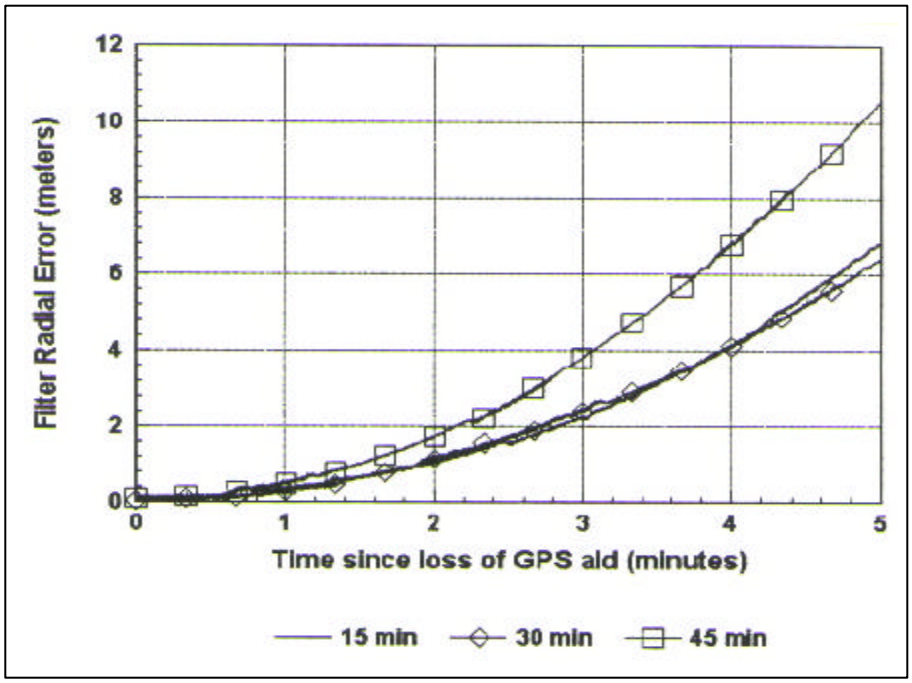


Figure 3.17: Filter Radial Error versus Time Since Loss of DGPS Aid – Worst Case.

CHAPTER 4
NOVATEL BEELINE GPS POSITIONING SYSTEM

The Novatel Beeline GPS system offers both real-time kinematic position (accurate to 25 cm CEP) and orientation data (accurate to 0.4 deg) at 5 Hz. The system costs approximately $25,000.

Unlike the Ashtech Z-12 GPS receiver which can process 12 channels of L1 (C-code, coarse code) and L2 (P-code, precise code), the Beeline uses a dual antenna, 16 channel L1/L1 system (see Figure 4.1). A primary antenna is used as the reference point from which outputted position data is based. The secondary antenna is used to compute the vehicle baseline vector, giving azimuth (yaw) and pitch values. To compute for roll, a second beeline system can be installed 90 degrees (in the x-y plane) from the primary beeline set-up.

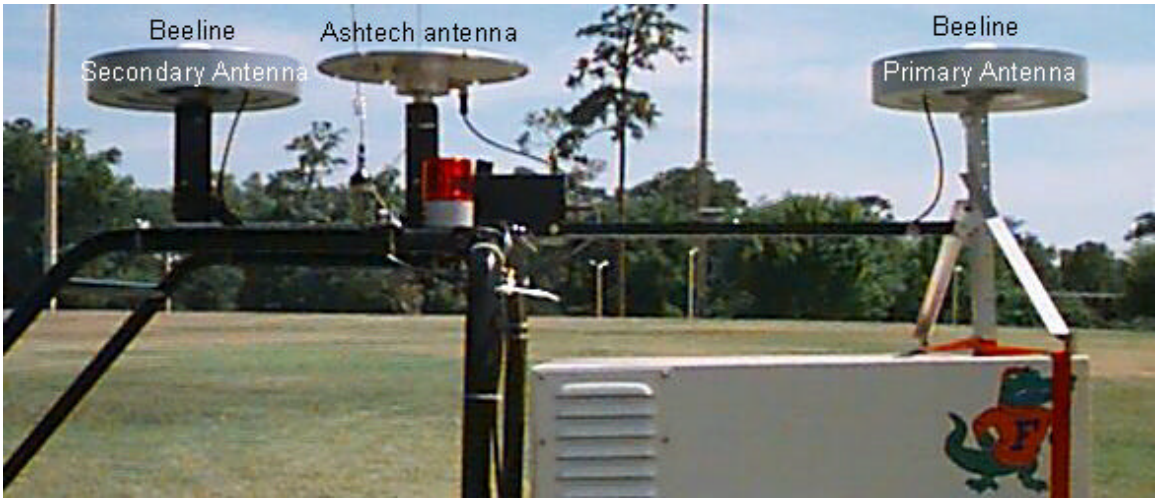


Figure 4.1: Novatel Beeline GPS Antenna Set-up on Navigation Test Vehicle.

Installation and Set-up

GPS Antennas

As described earlier, the Beeline system uses two antennas. Model 501 Novatel GPS antennas are mounted on choke rings (to lessen multipath signals that can result in erroneous position solutions). The vehicle baseline was set by aligning the primary and secondary antennas with the Ashtech antenna along the vehicle x-axis. This manual calibration will allow easy translation of either GPS system position point along the main vehicle axis. In order to accurately compare the output position of both the MAPS/POS and Beeline systems, the antenna offset and baseline distance were strictly measured to within 1 mm. The antenna offset (distance between Ashtech antenna and Beeline primary antenna) was measured as 0.957 m. The baseline distance (distance between Beeline primary and secondary antennas) was measured as 1.359 m. Novatel recommends at least a 1.0 m baseline distance to get good orientation values and avoid multipath errors. The baseline distance, which was then inputted as a fixed parameter in the Beeline receiver configuration, allows for faster attitude alignment and more accurate attitude solutions. Figure 4.2 shows the antenna configuration.

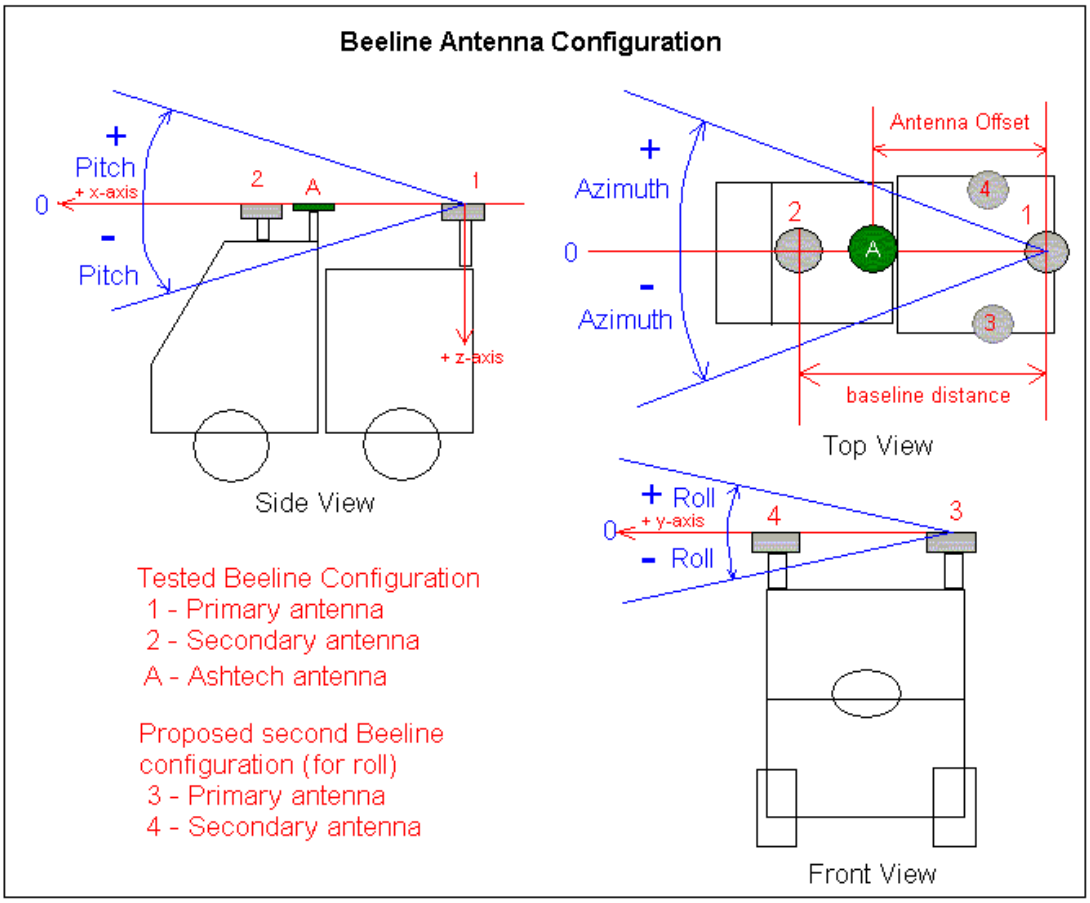


Figure 4.2: Beeline Antenna Configuration.

Beeline shelf

The Beeline shelf, shown in Figure 4.3, contains the Beeline receiver, a Freewave wireless data transceiver (radio), a PC104 computer stack, cables and connections, and a 12 VDC input power receptacle. A Null-modem serial data cable connects the second serial port on the PC104 to COM1 of the Beeline receiver. This is the main data line where position and orientation data are sent and processed by the beeline program on the PC104. The controlling program runs on a LYNX real-time operating system. A straight serial data cable connects COM2 of the Beeline receiver with the Freewave radio. This line is used for incoming differential corrections sent by the RT20 base station receiver.

The radio connects to a MAXRAD 9053 radio antenna. RF antenna cables connect RF1 and RF2 ports on the Beeline receiver to the primary (RF1) and secondary (RF2) antennas. +12 VDC is supplied through an AC-DC converter, powered by a gas generator and backed-up by an UPS.

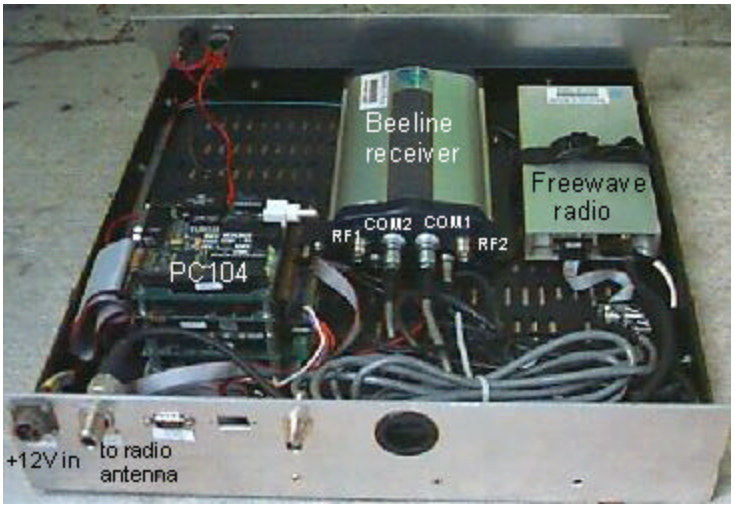


Figure 4.3: Beeline POS Shelf Layout.

Base Station

The Base Station is used to send differential corrections from a base station GPS receiver to the rover GPS receiver. The base point over which the base antenna will be located is a fixed pre-surveyed point. The existing base station point (which is used for all autonomous tests in Flavet field) was used a reference point to survey in a second base point. Using 2 Ashtech receivers, the original base point (base point 1) was set-up as the base and a second point was arbitrarily located approximately 6.3 m away due east. Using post-processing, the coordinates of the second point were calculated to millimeter (2 mm) accuracy. Tables 4.1 and 4.2 lists the exact coordinates and distances.

Table 4.1: Base Station Coordinates.

Position	Latitude (dec.deg)	Longitude (dec.deg)	Altitude (m)
Base Point 1	29.6472687	-82.35443972	15.000
Base Point 2	29.64726896	-82.35437489	14.8615

Table 4.2: Base Point Distances.

	Base 1 to Base 2
Northing Distance (m)	-0.0289
Easting Distance (m)	-6.272
Total Distance (m)	6.272

The base station antennas were set-up as follows: base point 1 – Ashtech ; base point 2 – Beeline (see Figure 4.4). The short separation allowed for all GPS receivers and radios to be housed in one portable case. This includes a Ashtech Z-12 GPS receiver connected to a Freewave radio (with a whip antenna), a Novatel RT-20 GPS receiver connected to a second Freewave radio (with a MAXRAD antenna), and a 12V AC-DC converter. Normal operation is powered by a Honda gas generator with a rechargeable battery pack as back-up. The base station unit is shown in Figure 4.5.

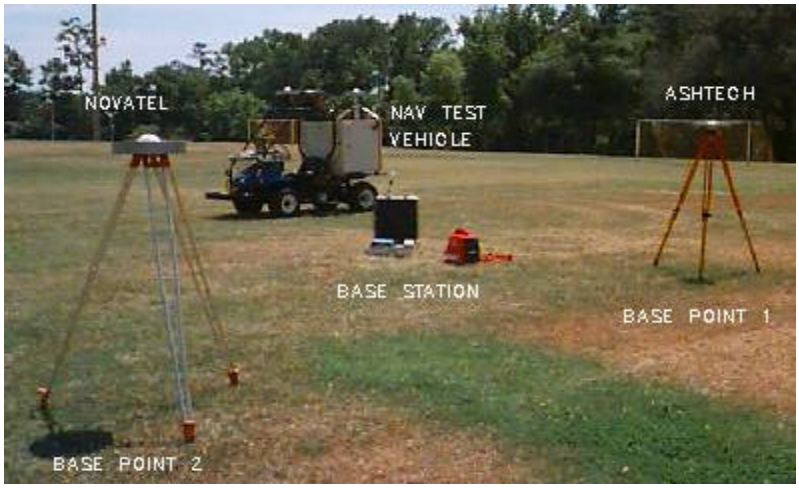


Figure 4.4: Base Station Set-Up on UF Flavet Field.

Figure 4.5: Dual Base Station GPS Receiver and Radio Unit.

Testing Procedures

The performance of the Beeline was measured relative to the MAPS/Ashtech system. All tests were conducted in Flavet field (commonly known as the bandshell) in the University of Florida. The tests were broken down into: (1) Alignment, (2) Accuracy, (3) Recovery, (4) Latency, (5) Weather. Due to persistent problems with the beeline receiver, the Latency and Weather tests were not performed.

Alignment test

The Alignment test is divided into a Static and Dynamic test. In the Static Align test, the vehicle was parked approximately 42 meters away from base point 2 for all runs. Runs lasted 30 minutes (to allow sufficient time for position and orientation to converge). The Dynamic Align test, which also lasted 30 minutes, was conducted by running the vehicle autonomously doing a minor field sweep. Data loggers were used by both systems to record all position and orientation data.

Accuracy Test

In the Static part of the accuracy test, the vehicle was parked in 4 set directions. Using the yaw value of the MAPS/Ashtech system, the vehicle was oriented as follows: (1) Due North – 0 degrees, (2) Due East – 90 degrees, (3) Due South – 180 degrees, (4) Due West – 270 degrees. The yaw value was zeroed in to ±0.1 degree. Data was gathered at each position for 5 minutes twice during each run. In the Dynamic accuracy test, the vehicle ran a long field sweep (1 hour), with both MAPS/Ashtech and Beeline systems collecting data in real-time.

Recovery Test

The recovery test consisted of position degradation due to 2 cases: (1) primary antenna losing satellites, and (2) differential corrections not being received by the Beeline. The first case was simulated by covering the primary antenna for fixed duration. The second case was achieved by breaking the radio link between base and rover for a specified amount of time.

Calculations, Data Recording, and Post-Processing

Both systems used separate data loggers to output position files. These were formatted to give data in the nearly the same format to facilitate post-processing. As explained earlier, the MAPS/Ashtech system reads in data from both GPS receiver and MAPS unit, computes a position and orientation solution through a Kalman filter. The output file is then sent to to Mobility Control Unit (MCU) which uses it for vehicle navigation. The MCU contains the data-logger for the position file. The output position is translated 0.957 m. along the negative vehicle x-axis to coincide with the measuring point of the primary Beeline antenna. This is accomplished by entering the offset value

in the POS configuration file. The output data screen of the position component on the MCU program is shown in Figure 4.6.

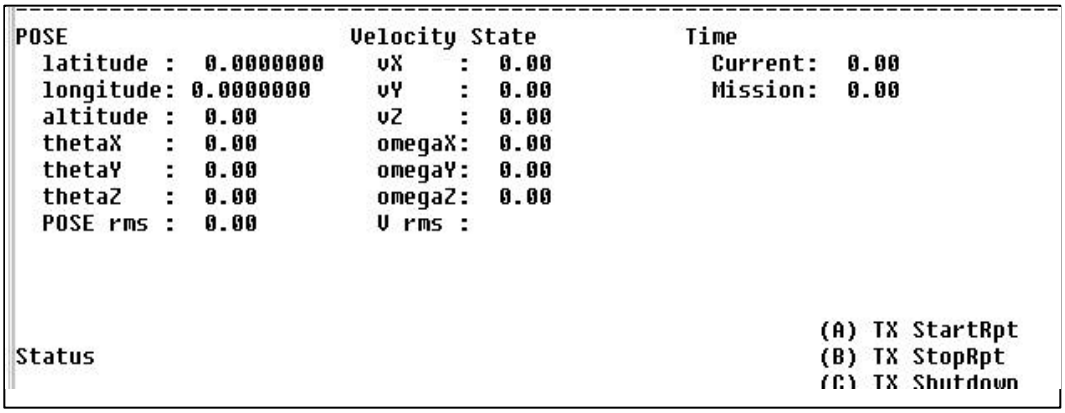


Figure 4.6: MAPS/Ashtech POS Output Data Screen on the MCU.

The Beeline uses 3 data logs, outputted by the Beeline receiver to the PC104 at 5 Hz. The PRTKA log contains the real-time kinematic position data in latitude, longitude and altitude. The ATTA log contains attitude (orientation) data in azimuth (yaw) and pitch. The VLHA log contains velocity data including horizontal speed over ground and track over ground (heading) [Nov98]. Accuracy of the data is characterized by error probabilities and denoted by the standard deviation values of the data fields. These values are also available in the logs. The output screen of the Beeline program is shown in Figure 4.7.

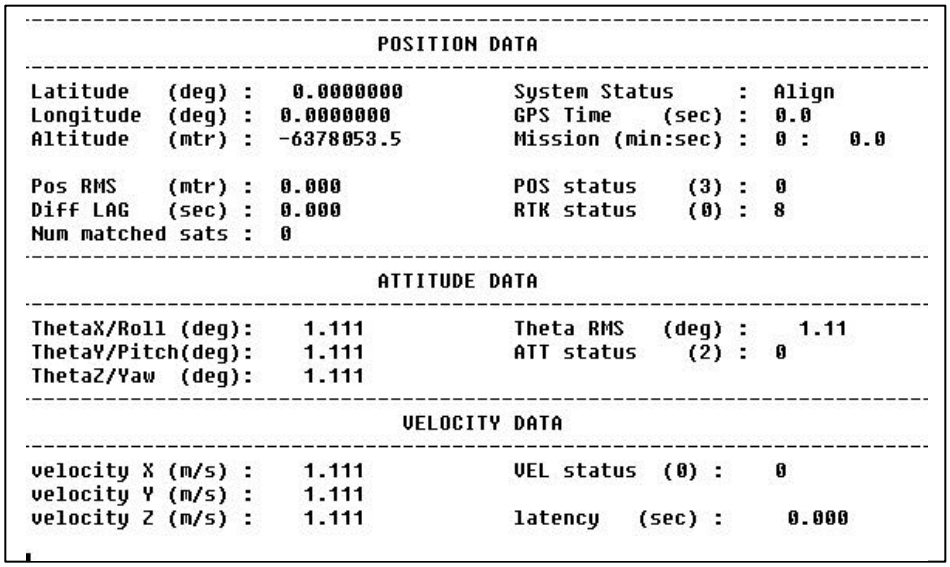


Figure 4.7: Beeline Data Output Screen.

The main benchmark output field of the Beeline program is the Pos RMS value.

$$\text{Pos RMS} = \sqrt{(\sigma_{\text{LAT}})^2 + (\sigma_{\text{LON}})^2 + (\sigma_{\text{ALT}})^2} \tag{3.1}$$

where σ_{LAT} is the latitude standard deviation

σ_{LON} is the longitude standard deviation

σ_{ALT} is the altitude standard deviation

The Pos RMS was used as the main accuracy measurement of the Beeline. Two other markers are the POS status and RTK status. POS status indicates which position mode it is in: (0) no position, (1) single point mode, (2) pseudorange differential mode, (3) RT-20 position. The RTK status indicates the type of position solution in RT-20 mode: (0) converged solution, (1) non-converged solution. Thus, for accurate real-time kinematic position, the POS status should read (3) and the RTK status should read (0). During preliminary testing, it was discovered that the RTK status (0) was achieved when the Pos RMS value dips to 0.5 m.

Similarly, for orientation, the Theta RMS (orientation error) was computed as

$$\text{Theta RMS} = \sqrt{(\sigma_{\text{AZIMUTH}})^2 + (\sigma_{\text{PITCH}})^2} \quad (3.2)$$

where σ_{AZIMUTH} is the azimuth standard deviation

σ_{PITCH} is the pitch standard deviation

The ATT status marker indicates the attitude type: (0) no attitude, (1) good 2D floating attitude solution, (3) good 2D integer attitude solution, (3) floating ambiguity solution with line bias known, (4) fixed ambiguity solution with line bias known.

For velocity calculation, the horizontal speed over ground is stripped into the individual components in the positive x, y, and z axes using the track over ground (velocity vector azimuth value) minus the current azimuth. The VEL status marker indicates the velocity state: (0) carrier phase based velocity, (1) Doppler data based velocity, (2) old Doppler velocity. VEL status (0) assures good velocity accuracy.

All data files were post-processed using a comparator program. This program reads in both the MAPS/Ashtech POS file and the Beeline data file, interpolates the GPS time stamps (to match the data), calculates position, orientation and velocity errors, and outputs 2 files, (1) 'out' file which contains the individual errors in time-based intervals, (2) 'stat' file which displays statistics of error values and data integrity. The 'out' file, which was used to plot the results, contains errors based on the following formula (MAPS/Ashtech data – Beeline data, e.g. Northing error = MAPS/Ashtech latitude – Beeline latitude). The 'stat' file summarizes the mean error and standard deviation error values.

The Position errors were calculated as (1) 3D error (rms error of latitude, longitude and altitude), and 2D error (rms error of latitude and longitude). The 3D error

gives the overall relative error between the 2 systems. However, of greater importance is the 2D error since presently, the vehicle uses only 2D path planning and vehicle control. Also, for most GPS systems, the altitude values have higher error tolerances. In the actual tests, the setting up of the base station antennas were performed using a plumb bob and centered on the base point markers. However, the actual antenna height was never measured (even for past position tests).

Test Results

Static Align Test

All three test runs resulted in consistent position and orientation convergence. There was a direct correlation with Beeline Pos RMS and Pos error (MAPS/Ashtech – Beeline). The summary of the four runs is listed in below in Tables 4.3 and 4.4.

Table 4.3: Static Alignment Test Mean Error Values.

Beeline		ERROR							
Time	Pos RMS	Pos	Orient	Vel	North	East	Alt	Pitch	Yaw
1734.8	0.089	0.079	2.048	0	0.005	0.079	-0.032	-0.613	1.954
1544.6	0.093	0.059	0.177	0	0.004	-0.059	0.049	0.093	-0.151
1762.4	0.375	0.096	2.35	0.002	0.061	-0.046	0.084	1.448	-1.426
5041.8	0.190	0.079	1.580	0.001	0.024	-0.007	0.033	0.324	0.128

Pos – 3D Position error (m)
Orient – 3D Orientation error (deg)
Vel – 3D Velocity error (m/sec)
North – Northing error (m)
East – Easting error (m)
Alt – Altitude (height) error (m)

Table 4.4: Static Alignment Test Standard Deviation Values.

Beeline		STANDARD DEVIATION (STDev)							
Time	Pos RMS	Pos	Orient	Vel	North	East	Alt	Pitch	Yaw
1734.8	0.089	0.111	1.596	0.049	0.055	0.097	0.114	0.486	1.52
1544.6	0.093	0.082	0.107	0.036	0.034	0.074	0.034	0.052	0.094
1762.4	0.375	0.072	2.49	0.056	0.035	0.062	0.042	1.833	1.667
5041.8	0.190	0.088	1.452	0.047	0.042	0.078	0.064	0.824	1.135

Pos – 3D Position error (m)
Orient – 3D Orientation error (deg)
Vel – 3D Velocity error (m/sec)
North – Northing error (m)
East – Easting error (m)
Alt – Altitude (height) error (m)

Table 4.5 gives a time-based summary of the error parameters during alignment. Initially, when the POS status reaches (3) – RT20 position, the Pos RMS value starts at 3.5-4.4 m which translates to 1.71 m actual 3D Pos error. This converges exponentially, and after 30 minutes reaches a Pos RMS of 0.172 m translating to 3D Pos error of 0.099 m. The 2D Pos error (lat+lon) was about 60-70% of the 3D error. Taking only 2D position, it takes 60 seconds to achieve position to within 0.3 m. of the MAPS/Ashtech. For 3D position, it takes 200 seconds to achieve 0.3 m accuracy. The average time to converge to RTK status (0) – (converged solution of 0.5 m. Pos RMS) is 259.8 sec. Through preliminary tests, it was determined that a 0.3 m Pos RMS gives adequate position accuracy (0.09-0.12 m). It took 614.4 sec to reach the 0.3 m benchmark. The attitude (orientation) solution converged after 837.2 sec. This process is referred to as line bias calibration. In cold boot-up, this normally takes 10-15 minutes. Once the line bias has been calibrated, accurate orientation is outputted.

Table 4.5: Static Alignment Test Time-Based Error Summary.

Time (sec)	Pos RMS (m)	3D Pos (m)	2D Pos (m)	Orient (deg)	North (m)	East (m)	Alt (m)	Pitch (deg)	Yaw (deg)	Remarks (status)
0	4.076	1.706	1.259	22.216	1.188	0.418	1.148	20.308	8.389	
60	1.168	0.871	0.266	7.623	0.152	0.142	0.792	5.868	3.218	
120	0.829	0.482	0.266	4.483	0.174	0.193	0.398	1.207	2.274	
180	0.645	0.31	0.216	6.963	0.131	0.132	0.218	3.443	5.772	
240	0.529	0.28	0.221	4.271	0.12	0.158	0.151	0.469	4.199	
259.8	0.5	0.294	0.238	3.619	0.115	0.188	0.152	0.557	3.555	converged position
300	0.45	0.268	0.206	3.274	0.078	0.187	0.163	0.758	3.164	
400	0.361	0.218	0.185	3.702	0.035	0.181	0.11	1.844	3.077	
500	0.326	0.107	0.095	3.694	0.033	0.087	0.049	2.399	2.459	
600	0.303	0.097	0.09	2.818	0.031	0.084	0.03	2.28	1.119	
614.4	0.3	0.094	0.091	2.793	0.033	0.085	0.022	2.309	1.051	good position
837.2	0.266	0.079	0.066	0.712	0.036	0.054	0.041	0.356	0.608	converged attitude
900	0.257	0.09	0.076	0.708	0.04	0.06	0.048	0.381	0.595	
1200	0.23	0.145	0.101	0.929	0.023	0.097	0.101	0.214	0.892	
1500	0.197	0.088	0.069	0.741	0.031	0.052	0.052	0.376	0.59	
1800	0.172	0.099	0.066	0.354	0.032	0.056	0.069	0.256	0.202	

The results of Table 4.5 for Pos RMS, 3D Pos error, 2D Pos error are shown in Figure 4.8. It is clear that the 3D Pos error is always less than the Beeline RMS. This accounts for the fact that the error of the filter position solution has not been factored in. Figure 4.9 shows that during the first minute of alignment, the northing error is greater than the easting error. Afterwhich, the easting error converges to a higher value (0.056 m) versus the northing error (0.032 m). The altitude error was higher than both northing and easting all thoughout the alignment process and converges to 0.069 m. In Figure 4.10, orientation error shows no clear pattern as both pitch and azimuth errors fluctuated.

However, both converged to error values of 0.2-0.25 deg after 30 minutes.

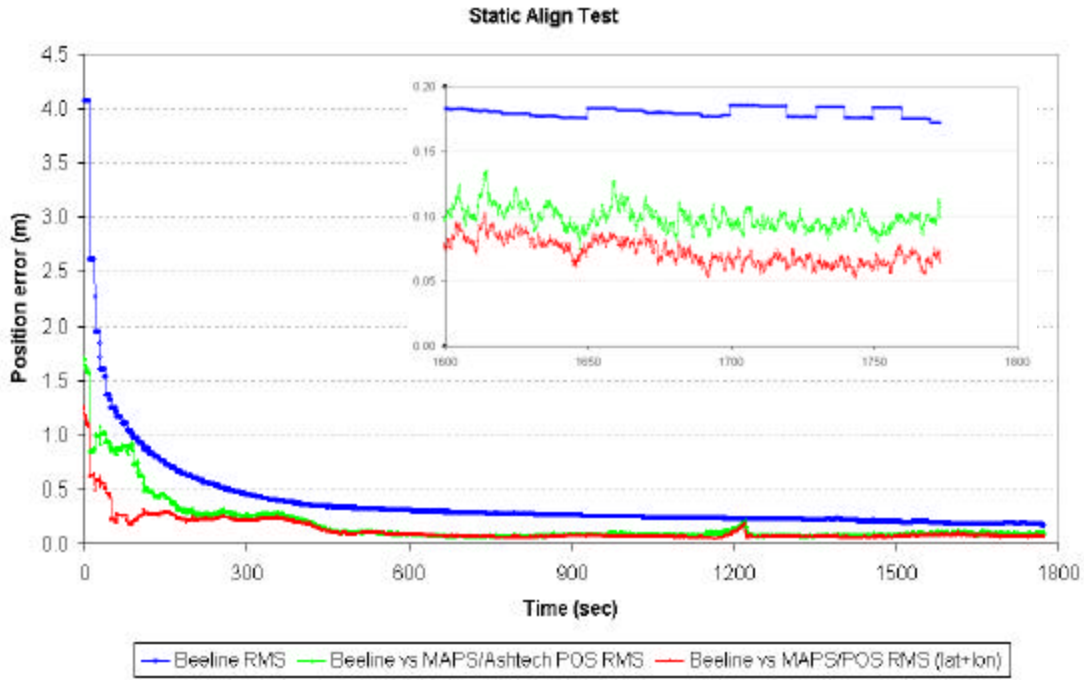


Figure 4.8: Static Alignment Test Overall Position Error.

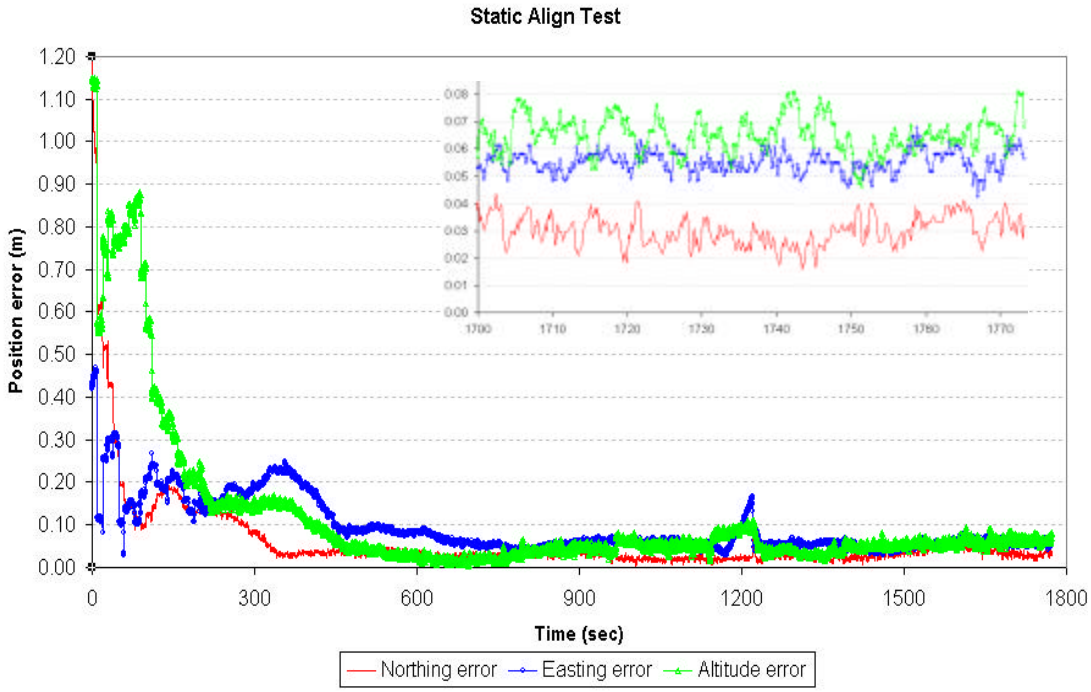


Figure 4.9: Static Alignment Test Directional Position Error.

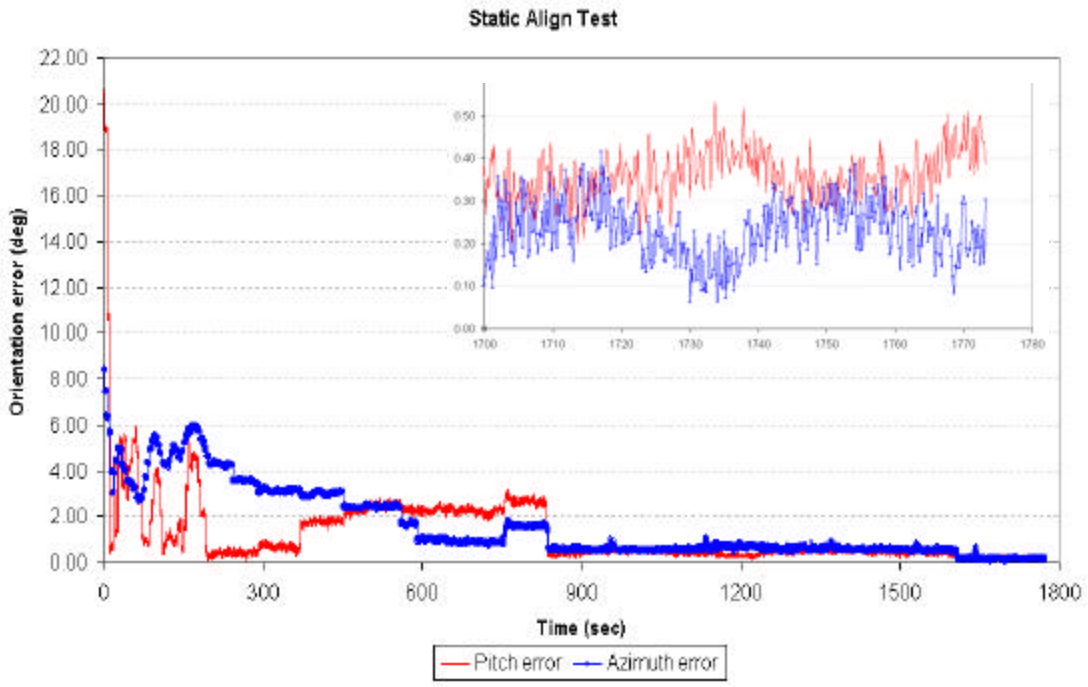


Figure 4.10: Static Alignment Test Orientation Error.

Dynamic Align Test

Similarly, all three test runs resulted in consistent position and orientation convergence. There was a direct correlation with Beeline Pos RMS and Pos error with convergence being slower than in static mode. It should be noted that during alignment, the line bias has already been determined and thus, good attitude was already available. The overall mean POS rms and 2D Pos error were higher for the dynamic mode than the static mode. However, the standard deviation error values did not show significant difference between dynamic and static modes. Velocity error values were low in the order of 0.03±0.1 m/sec. The summary of the three runs is listed in Tables 4.6 and 4.7.

Table 4.6: Dynamic Alignment Test Mean Error Values.

RUN		MEAN ERROR										
Time (sec)	PosRMS (m)	2D Pos (m)	Orient (m)	Vel (m)	North (m)	East (m)	Alt (m)	Pitch (deg)	Yaw (deg)	Vel X (m/s)	Vel Y (m/s)	Vel Z (m/s)
870	0.13	0.178	0.305	0.022	0.035	-0.175	-0.062	0.171	0.253	-0.013	0.003	-0.018
1343.8	0.33	0.188	0.207	0.028	0.174	-0.072	-0.086	0.054	0.2	-0.011	0.002	-0.026
2354.2	0.27	0.046	2.993	0.035	-0.011	0.023	0.112	2.918	0.24	-0.004	-0.001	-0.033
4568.0	0.261	0.113	0.246	0.030	0.052	-0.043	0.021	0.100	0.221	-0.008	0.001	-0.028

Table 4.7: Dynamic Alignment Test Standard Deviation Values.

RUN		STANDARD DEVIATION (STDev)										
Time (sec)	PosRMS (m)	2D Pos (m)	Orient (m)	Vel (m)	North (m)	East (m)	Alt (m)	Pitch (deg)	Yaw (deg)	Vel X (m/s)	Vel Y (m/s)	Vel Z (m/s)
870	0.13	0.088	0.44	0.105	0.032	0.082	0.1	0.201	0.391	0.045	0.079	0.053
1343.8	0.33	0.16	0.391	0.12	0.092	0.131	0.092	0.107	0.377	0.049	0.085	0.07
2354.2	0.27	0.054	4.162	0.157	0.045	0.03	0.089	1.008	4.011	0.051	0.135	0.062
4568.0	0.261	0.092	2.344	0.136	0.056	0.070	0.092	0.589	2.253	0.049	0.110	0.063

In Table 4.8, position convergence is tabulated in time-based form. It almost twice as long for the position solution to converge (dynamic - 542.0 sec vs. static – 259.8 sec). This was also the case in reaching good position (0.3 m Pos RMS) - (dynamic – 1083.4 sec vs. static - 614.4 sec). It can also be seen that even without a converged position solution, it takes 300 sec (5 minutes) to reach reasonably good 2D position (<0.2 m error). Pos RMS converges exponentially, but is greatly influenced by the number of satellites in view. For all tests, 6-8 satellites were visible 95% of the time. Figures 4.11, 4.12, and 4.13 show Overall Position, Directional Position and Orientation error curves over time. It should be noted that in dynamic mode, the altitude error converged similarly as in static mode (to 0.06-0.07 m), but the northing and easting errors converged to a low value of 0.02 m.

Table 4.8: Dynamic Alignment Test / Time-Based Error Summary.

Time (sec)	Pos RMS (m)	3D Pos (m)	2D Pos (m)	Northing (m)	Easting (m)	Altitude (m)	Remarks (status)
0	4.901	1.161	0.415	0.191	0.368	1.084	
60	1.316	1.018	0.695	0.588	0.46	0.861	
120	1.036	0.611	0.443	0.426	0.353	0.432	
180	0.909	0.745	0.442	0.316	0.158	0.484	
240	0.819	0.495	0.332	0.229	0.182	0.331	
300	0.752	0.277	0.192	0.097	0.181	0.142	
400	0.671	0.333	0.234	0.169	0.16	0.162	
500	0.542	0.278	0.224	0.161	0.117	0.18	
542	0.5	0.305	0.21	0.133	0.133	0.168	Converged position
600	0.452	0.245	0.233	0.146	0.131	0.098	
900	0.338	0.179	0.137	0.092	0.07	0.067	
1083.4	0.3	0.174	0.111	0.091	0.056	0.062	good position
1200	0.281	0.145	0.099	0.081	0.065	0.022	
1500	0.308	0.164	0.123	0.018	0.067	0.047	
1800	0.249	0.106	0.047	0.015	0.014	0.051	

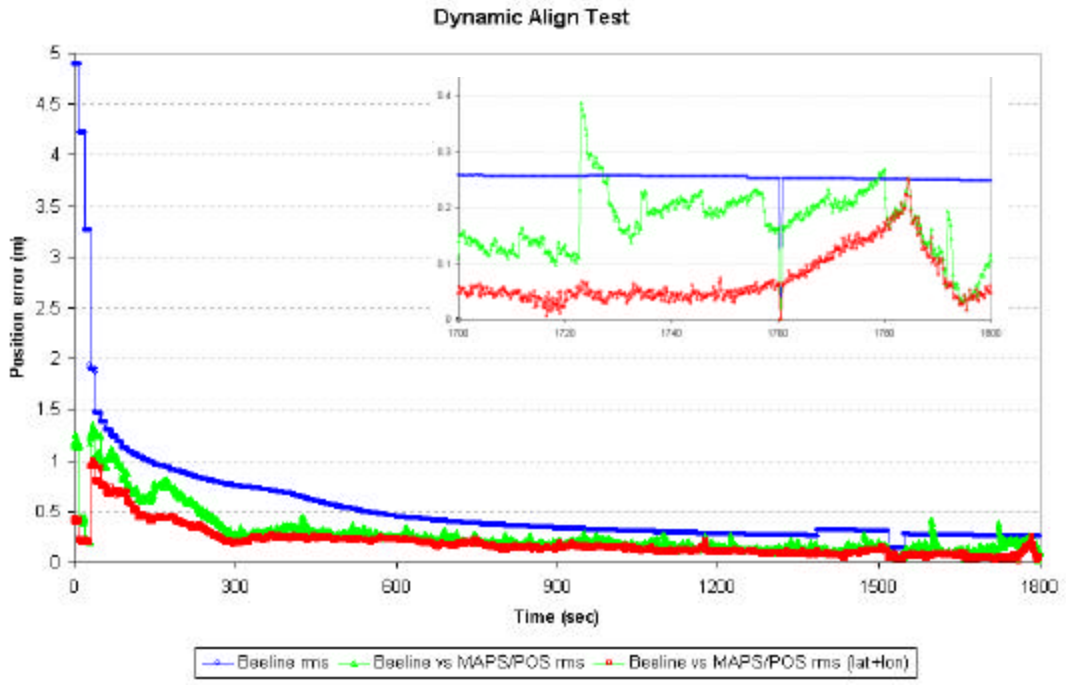


Figure 4.11: Dynamic Alignment Test Overall Position Error.

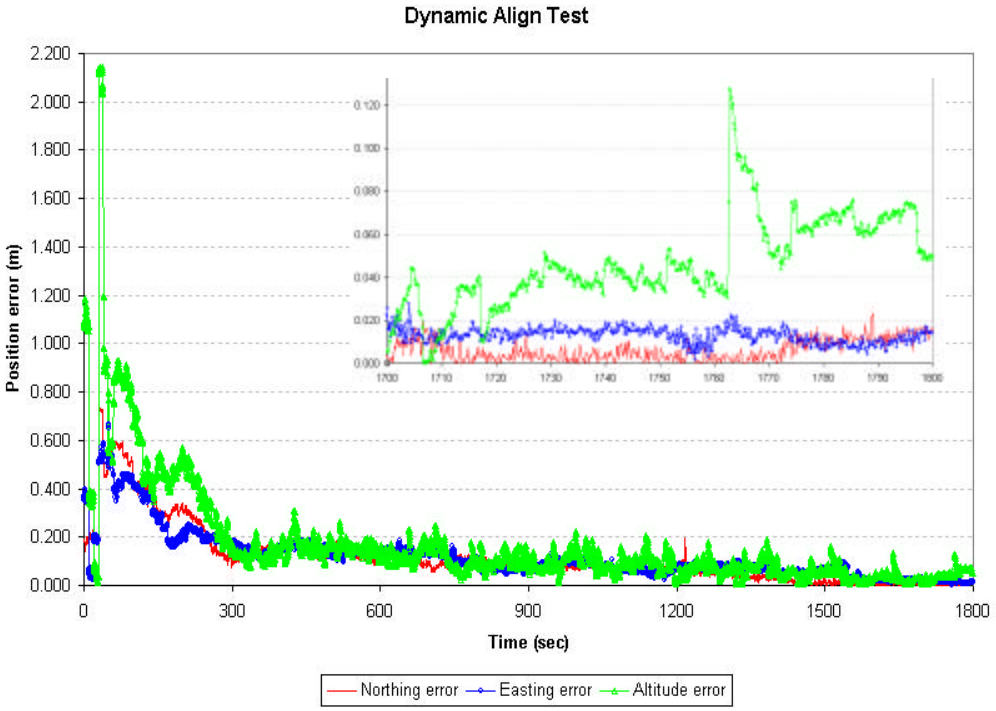


Figure 4.12: Dynamic Align Test Directional Position Error.

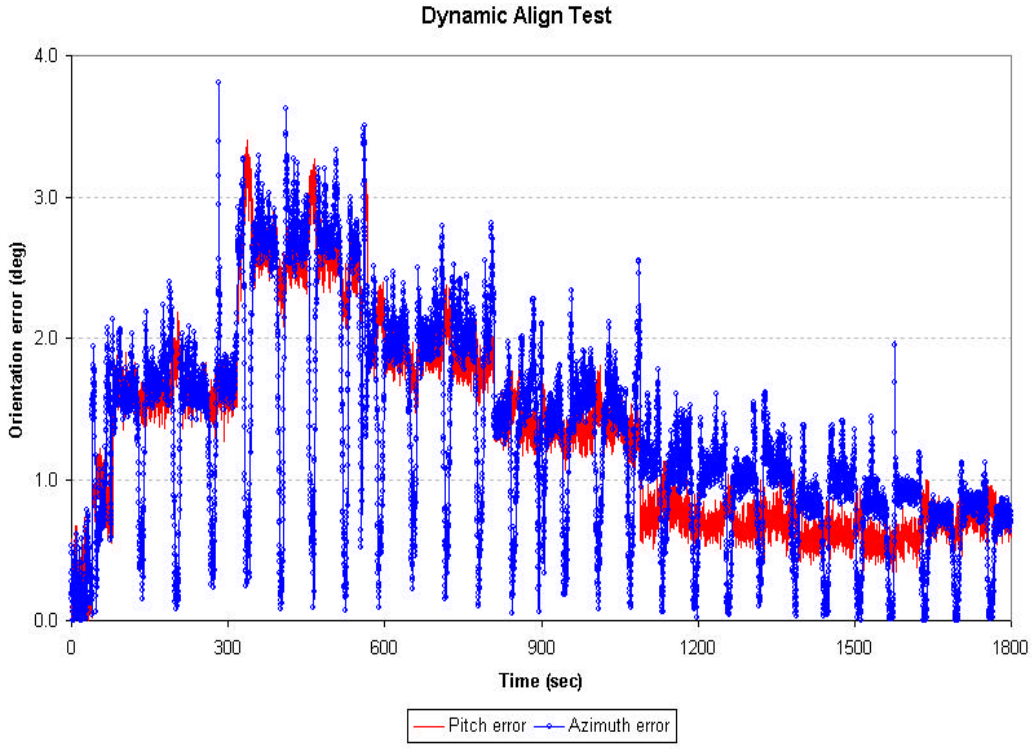


Figure 4.13: Dynamic Alignment Test Orientation Error.

Static Accuracy Test

In performing this test, the Pos RMS was converged to 0.3±0.05 m. The mean values for both runs shows that for a Pos RMS of 0.291, the mean 2D Pos error is 0.058 m and mean orientation error value of 0.4 deg. Standard deviation values for most datafields were lower due to static operation and already low Pos RMS (converged). The results are summarized in Tables 4.9 and 4.10.

This test is a good measure of the antenna calibration. By looking at northing and easting error values as heading changes in Table 4.11, the existence of offset distances can be pinpointed. In this case, no pattern was evident since northing and easting errors were consistent as the vehicle was rotated. This means that the relative to the MAPS/Ashtech position, the Beeline converged from the same direction for both runs. Further testing may reveal this behavior to be consistent and thus, can be anticipated and even corrected.

Table 4.9: Static Accuracy Test Mean Error Values.

RUN		MEAN ERROR							
Time (sec)	PosRMS (m)	2D Pos (m)	Orient (m)	Vel (m)	North (m)	East (m)	Alt (m)	Pitch (deg)	Yaw (deg)
1430.2	0.242	0.047	0.203	0.001	0.011	-0.046	0.015	0.182	0.091
1155.8	0.352	0.071	0.645	0.002	0.056	-0.003	0.089	0.12	-0.045
2586.0	0.291	0.058	0.401	0.001	0.031	-0.027	0.048	0.154	0.030

Table 4.10: Static Accuracy Test Standard Deviation values.

RUN		STANDARD DEVIATION (STDev)							
Time (sec)	PosRMS (m)	2D Pos (m)	Orient (m)	Vel (m)	North (m)	East (m)	Alt (m)	Pitch (deg)	Yaw (deg)
1430.2	0.242	0.053	0.243	0.041	0.049	0.022	0.035	0.142	0.197
1155.8	0.352	0.059	1.04	0.059	0.028	0.052	0.038	0.837	0.551
2586.0	0.291	0.056	0.599	0.049	0.040	0.035	0.036	0.453	0.355

Table 4.11: Static Accuracy Test Directional Position Error.

Heading	ERROR										
Due	Pos RMS	3D Pos	2D Pos	North	East	Alt	Orient RMS	ATT RMS	Ref. Yaw	Pitch	Yaw
North	0.333	0.111	0.085	0.013	-0.040	0.069	0.375	0.783	0.012	0.161	0.143
East	0.324	0.086	0.058	0.023	-0.030	0.054	0.251	0.760	90.012	-0.013	0.137
South	0.319	0.078	0.064	0.033	-0.023	0.015	0.374	0.770	180.037	0.037	-0.015
West	0.302	0.084	0.065	0.037	-0.022	0.041	0.294	0.723	270.051	-0.065	0.037

Dynamic Accuracy Test

In performing this test, the runs had to be of sufficient duration with the GPS Pos RMS reamining at a good level. The two runs were performed at 2 different Pos RMS levels (Pos RMS<0.2m, Pos RMS<0.4 m.). As shown in Tables 4.12 and 4.13, as expected, the first run, with a mean Pos RMS of 0.186 m. showed good position accuracy (mean 2D Pos error = 0.044 m) The second run, with Pos RMS of 0.312 m. had a higher mean 2D Pos error of 0.069 m. Mean orientation error was slightly better for run 1 while mean velocity error was comparable. Standard deviation values did not show significant difference for both runs.

Figure 4.14 shows the autonomous field sweep with both system 2D position traces (blue – MAPS/Ashtech; red – Beeline). Arrows indicate direction of the field sweep. In Figure 4.15, the minor position convergence during sweep is characterized. The individual positional errors (northing, easting and altitude) are shown in Figure 4.16. It can be clearly seen that vehicle yaw has an effect on error, especially altitude. Lastly, Figure 4.17 shows orientation error over time (and as a function of vehicle yaw – yellow line).

Table 4.12: Dynamic Accuracy Test Mean Error and Standard Deviation.

RUN		MEAN ERROR										
Time (sec)	PosRMS (m)	2D Pos (m)	Orient (m)	Vel (m)	North (m)	East (m)	Alt (m)	Pitch (deg)	Yaw (deg)	Vel X (m/s)	Vel Y (m/s)	Vel Z (m/s)
2963.6	0.186	0.044	0.268	0.027	0.033	0.029	0.071	-0.014	0.267	-0.011	0.001	-0.025
2307.8	0.312	0.069	0.448	0.024	0.003	0.066	0.035	0.389	0.219	-0.009	0.001	-0.023
5271.4	0.241	0.055	0.347	0.026	0.020	0.045	0.055	0.162	0.246	-0.010	0.001	-0.024

Table 4.13: Dynamic Accuracy Test Standard Deviation Values.

RUN		STANDARD DEVIATION (STDev)										
Time (sec)	PosRMS (m)	2D Pos (m)	Orient (m)	Vel (m)	North (m)	East (m)	Alt (m)	Pitch (deg)	Yaw (deg)	Vel X (m/s)	Vel Y (m/s)	Vel Z (m/s)
2963.6	0.186	0.045	0.316	0.119	0.021	0.04	0.079	0.107	0.297	0.047	0.086	0.068
2307.8	0.312	0.054	0.276	0.11	0.032	0.043	0.1	0.094	0.259	0.043	0.079	0.064
5271.4	0.241	0.049	0.298	0.115	0.026	0.041	0.088	0.101	0.280	0.045	0.083	0.066

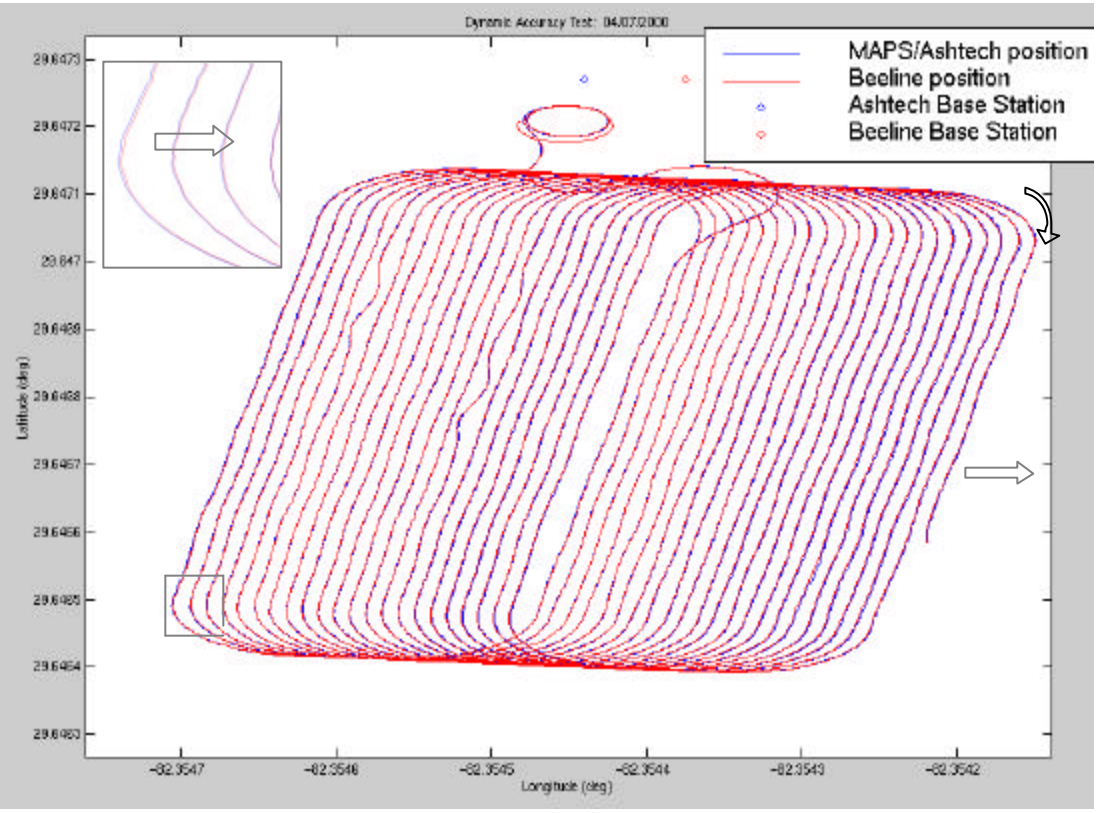


Figure 4.14: Dynamic Accuracy Test Autonomous Field Sweep.

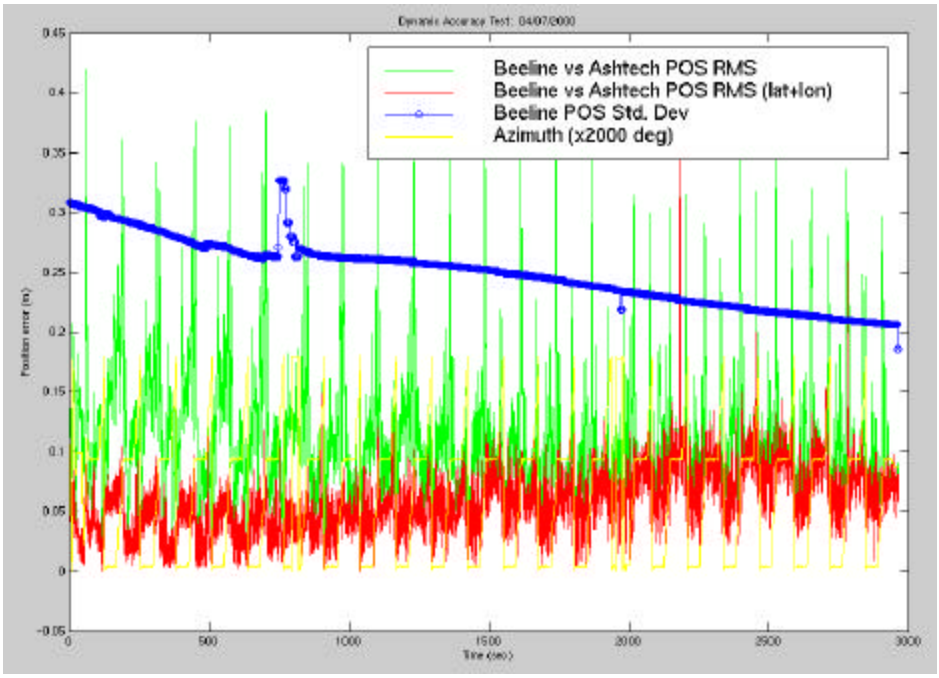


Figure 4.15: Dynamic Accuracy Test Position Error.

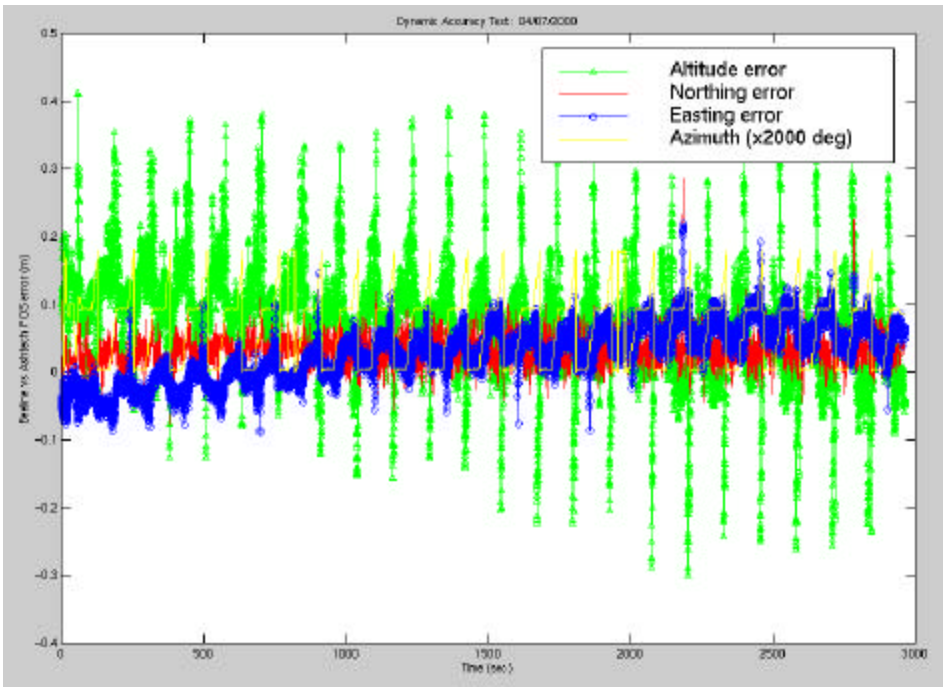


Figure 4.16: Dynamic Accuracy Test Directional Position Error.

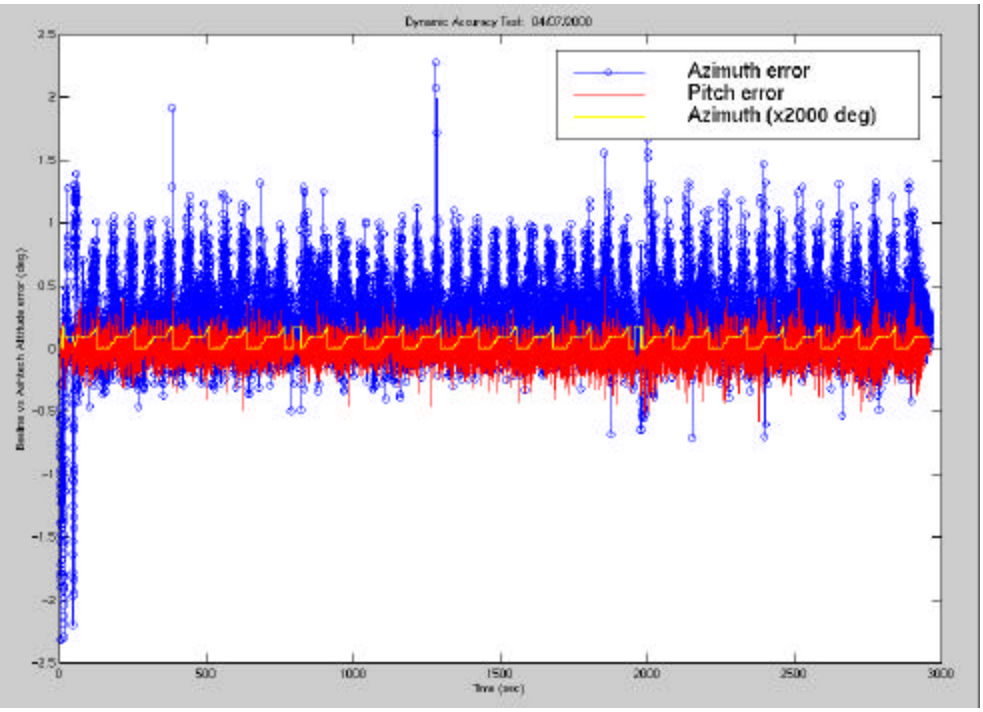


Figure 4.17: Dynamic Accuracy Test Orientation Error.

Recovery Test

As described earlier, GPS loss will be simulated artificially by (1) Covering Primary antenna and (2) Interrupting Transmission of Differential Corrections from Base. Covering the primary antenna simulates GPS loss when the vehicle encounters situations when direct satellite lock is comprised (like going underneath trees). By interrupting transmission of differential corrections, several situations are simulated such as bad radio link, vehicle going out of radio range, and temporary loss of GPS lock at the base station.

The position convergence of the Beeline must be reestablished. The output position solution of the receiver goes through 4 phases.

Phase 1 : Pos Status (0) – No position, when locked satellites in view are less than 4.

Phase 2 : Pos Status (1) – Single Point; ≥ 4 locked satellites; Pos RMS = 120 – 50 m.

→ *Differential Corrections available*

Phase 3 : Pos Status (2) – Pseudorange Differential; ≥ 4 satellites; Pos RMS = 4 – 1.6 m.

Phase 4 : Pos Status (3) – RT-20 position; ≥ 4 satellites; Pos RMS = 5.5 – 0.04 m.

During normal operation, cold-boot alignment (initially turn on base station receiver, then rover receiver) takes approximately 360 ± 20 sec to reach a converged solution (Pos RMS of 0.5 m). Based on the static alignment test data, convergence while in Pos status 3 takes 260 sec. Thus, it takes 100 ± 20 sec to go from Pos status 0 to 3 during cold boot. Table 4.14 shows the mean times to Pos status convergence.

1. Covering Primary Antenna

Covering the primary antenna resulted in the primary antenna losing satellite lock. Once the number of satellites drops below 4, no position is reported (Pos status 0). This occurred almost instantaneously with total antenna blockage (within 2 sec). This case would be similar to the primary GPS antenna cable being disconnected. Once the cover has been removed, it takes 18-23 sec for reacquisition of the satellites and subsequent outputting of a single point position solution. From this point, the convergence follows that described in Table 4.14.

Table 4.14: Receiver Cold Boot.

Pos Status	Time (sec)	Remarks
0 → 1	20.7 ± 0.4	Must have at least 4 satellites in view
1 → 2	38.3 ± 5.2	Upon reception of differential corrections
2 → 3	43.5 ± 8.5	When RT-20 position is computed
0 → 3	102.5 ± 4.1	Total time to RT-20 position
3 → 0.5 m Pos RMS	259.8	RT-20 position convergence (carrier-phase)

2. Interrupting Transmission of Differential Corrections

Differential corrections were suppressed by turning off the radio at the base station end. To accurately follow the position solution degradation, the radio link was suppressed for different time intervals. In RTK mode, the latest low-latency position solution is reflected for up to 30 seconds after the reception of the last base station observations. This means that withholding transmission of differential corrections for over 30 seconds will result in the position solution reverting from RT-20 mode to either single point or pseudorange differential mode. Thus, four time intervals were selected (15 sec, 20 sec, 25 sec, and 30 sec). Table 4.15 summarizes the degradation of the Pos RMS with increasing differential lag (time since last reported base station observations). Figure 13 clearly shows the similar pattern of degradation regardless of the Pos RMS value at the time of interruption.

Also, it should be noted that the Pos RMS values converge to about the same value. After 30 seconds, the Pos RMS value degrades to 2.7-2.9 m. However, as soon as a new set of differential corrections is received within 30 seconds, the Pos RMS reverts back to the initial value (before the lag). It should be also noted that the base station receiver was set up to output new satellite observation information every second. Thus, once the differential lag exceeds a second, there is interruption in the base-rover receiver link. As soon as the differential lag goes past 30 seconds, the position solution reverts from Pos status 3 (RT20) to Pos status 1 (single point). Once radio link has been reestablished, corrections are instantaneously processed and Pos status jumps to 2 (pseudorange). After 3-5 sec, position solution improves from Pos status 2 to 3 (RT20). The amount of time past 30 seconds that the corrections have been compromised has a minor effect on the

time it takes the receiver to revert back to RT20 position.

Table 4.15: Recovery Test Pos RMS Degradation.

Diff. Lag	Beeline Pos RMS (m)				
sec	10 sec	15 sec	20 sec	25 sec	30 sec
0	0.491	0.467	0.530	0.779	1.071
2	0.492	0.468	0.531	0.779	1.071
4	0.501	0.474	0.537	0.782	1.073
5	0.512	0.493	0.543	0.786	1.076
6	0.53	0.513	0.555	0.792	1.079
8	0.594	0.582	0.597	0.816	1.094
10	0.702	0.616	0.671	0.859	1.125
12		0.732	0.784	0.929	1.170
14		0.886	0.936	1.029	1.242
15			1.027	1.093	1.288
20			1.612	1.533	1.644
25				2.171	2.177
30					2.887

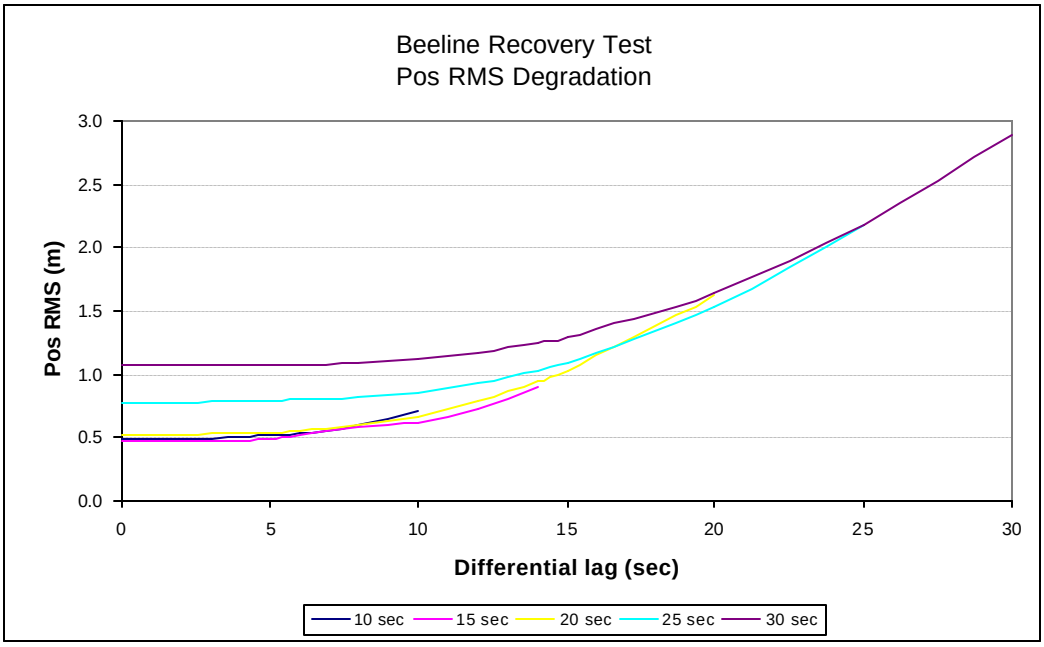


Figure 4.18: Recovery Test Interruption of Differential Corrections.

Comparatively, the Ashtech Z-12 GPS receiver takes a much faster time to converge. Since, the Z-12 is an L1/L2 channel receiver, it can converge to as low as 0.02-0.03 m RMS in 180-240 seconds (cold boot), depending on satellite availability. Applying the same radio link interrupting procedure, Table 4.16 shows the recovery times for different interruption times.

Table 4.16: Ashtech Z-12 GPS Receiver Recovery Data.

Differential Lag (sec)	Recovery time (sec)	RMS drop (m)
10	3	0.2 → 0.03
30	7	2.7 → 0.03
60	9	--
120	70	--

Analysis

The main objectives were to evaluate the system performance of the Novatel Beeline GPS system and compare it to the MAPS/Ashtech Position system under normal and less than ideal operating conditions. It has been established, based on manufacturer performance specifications, that the Beeline is a less accurate system for measuring position and orientation. Using the MAPS/Ashtech Pos as the reference system, position and orientation differences (errors) were calculated while performing specific tests.

The first test involved static alignment. Because the MAPS needs to be aligned at startup, initialization of the MAPS/Ashtech Pos system takes 10-16 minutes until Ready status is reached and the Pos RMS drops to 0.22-0.28 m. This Pos RMS value is not an absolute value and should only be taken as a relative indicator of the goodness of solution. The Beeline, on the other hand, takes a longer time to align. It takes about 6 minutes to reach a converged position solution. However, for better results, and

additional 6 minutes drops the Pos RMS to 0.3 m (with a relative mean error of 9.4 cm). Orientation alignment necessitates line bias calibration which takes 18-20 minutes from start-up. These long alignment times make using the Beeline disadvantageous when GPS performance is compromised .

As expected, the Beeline Pos RMS converged to a value of 0.18-0.20 m. When comparing this to the MAPS/Ashtech position, a mean error of 0.10 m. is observed. This means that the MAPS/Ashtech position is between the true position and Beeline position. It must be remembered that these error and RMS values are radial (circular) errors. With the MAPS/Ashtech position converged, the Beeline approaches known vehicle position to within 10 cm. Orientation values also approach known values to 0.3-0.5 deg (as specified).

Dynamic alignment becomes a factor when there is degradation in the position solution. Unlike the Ashtech Z-12, the Beeline receiver takes 3 to 6 times longer to converge. Thus, when differential corrections or satellite lock are compromised, the Beeline undergoes realignment, making accurate position availability longer. However, with integration to an inertial measurement unit, the Beeline can be a vital part of low cost, mid-level accuracy position system.

One important factor was antenna calibration. The vehicle axes were the main alignment tool in setting up the antennas in order to make accurate output position comparison possible. Great care should be taken in calibration. Performing an initial static accuracy test allows one to check for consistent antenna offsets overlooked during initial calibration. Here, no significant offsets were determined. Base station antenna location is also crucial. Locating the antennas consistently each time increases

differential position accuracy. Altitude, which is the least accurate position component, should also be measured carefully. Since the NTV presently uses only 2D path planning and navigation, base station altitude was never strictly monitored. Thus, the 2D position errors showed more consistent behavior.

During field sweep, dynamic accuracy seemed to be affected by current azimuth (yaw). 3D Pos errors were highest and lowest during turns into the longest straight portions of the path. This was due to the exact same behavior of altitude in these zones. As stated earlier, altitude is the least accurate of the position components and is thus susceptible to larger variations. 2D Pos errors were less affected by yaw. However, straight navigation did seem to decrease 2D Pos error. The sensitivity of the position data to the yaw value may be attributed to the behavior of the yaw value itself. Yaw error values were greatest during turning. There may be two reasons for this: (1) time lags in the receiver data which may cause mismatch, (2) greater dynamics are experienced and thus less accurate orientation solutions. Pitch, on the other hand, was not significantly affected by vehicle yaw or heading.

Recovery tests showed how critical satellite visibility and differential corrections reception are. Good GPS position and orientation is normally achieved with 6-8 satellites locked and tracked. Operating with 4-5 satellites not only slows down alignment but also presents a greater risk of dropping to a no position status. It was shown that covering the primary antenna resulted in losing position and resetting to the normal alignment process once antenna view is reestablished. For continuous real-time kinematic positioning, a solid differential correction link must exist. Temporary loss of base corrections leads to exponential degradation of the position accuracy and eventually to going out of carrier

phase differential mode. For the Beeline system, a maximum of 30 seconds of differential lag is allowed before position solution is compromised.

It should be noted that during testing, the Beeline receiver behaved inconsistently in maintaining position status. Throughout the entire testing period, the position solution would instantaneously degrade from RT20 (3) to pseudorange (2), therefore resetting the alignment process. This behavior was later found out to be signaled by an uncharacteristic jump in the differential lag without apparent disruption in the corrections transmission. The Beeline was able to hold its RT20 position up to 90 minutes, but with unpredictable occurrences of the status resetting. Thus, only limited good data was collected.

Novatel engineers confirmed that the integrity of the position solution was greatly affected by the solution rates. The RT-20 position update rate can go up to 5 Hz maximum. However, at this speed, only a single binary log can be read without expecting system failure. Since three separate ASCII data logs were read in, this was putting computational stress on the remote RT20 receiver. To eliminate costly system resets, a lower data rate (2 Hz) outputting binary logs should be used.

CHAPTER 5
HONEYWELL HG1700AG11 IMU / NOVATEL RT20 GPS POSITIONING SYSTEM

The exceptional performance of the Honeywell H-726 MAPS / Ashtech Z-12 GPS integrated positioning system comes at a high cost. A second alternative is to utilize the same concept of an INS/GPS integrated positioning system but with lower cost components. The selection of the components hinges on three factors: (1) performance, (2) cost, and (3) ease of integration. All three factors have to be weighed against the position system specifications and requirements needed as implemented on the NTV.

Overall System Description

The Inertial Navigation System (INS) is generally the most expensive and complex system component. The high cost of an INS is due mainly to the integration of inertial sensors (gyroscopes and accelerometers) with data processors to output position and orientation data already in navigation coordinates. Choosing a bare Inertial Measurement Unit, a Honeywell HG1700AG11, and developing the data processors, significantly reduces the INS cost. It must be noted, though, that the inertial sensors used by the H-726 MAPS are more accurate and precise than those of the HG1700AG11.

The GPS receiver component selection is based first on its real-time kinematic position accuracy. It has been established through previous testing and application that a positional accuracy of less than 0.3 m is still acceptable for ground vehicle navigation. Also, extensive

experience and familiarity are major reasons for using a Novatel RT-20 GPS receiver. In 1996, Dean Hutchinson verified the performance of the RT-20 using a train track configuration. Comparatively, the Ashtech Z-12 receiver was also tested and the results are in Table 5.1 [Hut97].

Table 5.1: Train Track Testing Summary (Novatel vs Ashtech).

Pos Error Distance (cm)	Novatel RT-20	Ashtech Z-12
Maximum	53.3289	78.1835
Minimum	0.2535	0.1477
Median	10.0218	8.7656
Mean	10.4141	9.9312
Std Deviation	5.2478	6.5358

The integration of the IMU and the GPS involves six steps:

(1) IMU Interfacing (data reception, error checking and averaging)

(2) IMU Calibration

(3) IMU Initial Alignment

(4) IMU Navigation Solution

(5) External Kalman Filter (GPS aiding)

(6) POS Message Output and Interface

Each step will be discussed in more detail in the succeeding sections.

Honeywell IMU

The Honeywell HG1700AG11 Inertial Measurement Unit is the low-cost solution for tactical weapon applications. This advanced IMU is designed to meet a need for a low-cost,

lightweight avionics system for installation in missiles, standoff weapons, torpedoes and unmanned aerial vehicles.

This strapdown IMU integrates the sensors and electronics, through the use of a common dither approach, to minimize redundant physical and electrical interfaces. The result is smaller size and reduced assembly cost.

System Specifications and Features

Specifically, the HG1700AG11 is designed to minimize cost through the use of:

(1) Miniature, low-cost tactical ring laser gyro (GG 1308 RLG)
(2) AlliedSignal RBA-500 accelerometers
(3) Standard modules/components across a wide range of applications
(4) Common dither approach
(5) Integral isolation ring

A block diagram of the system is shown in Figure 5.1. System specifications are listed in Table 5.2.

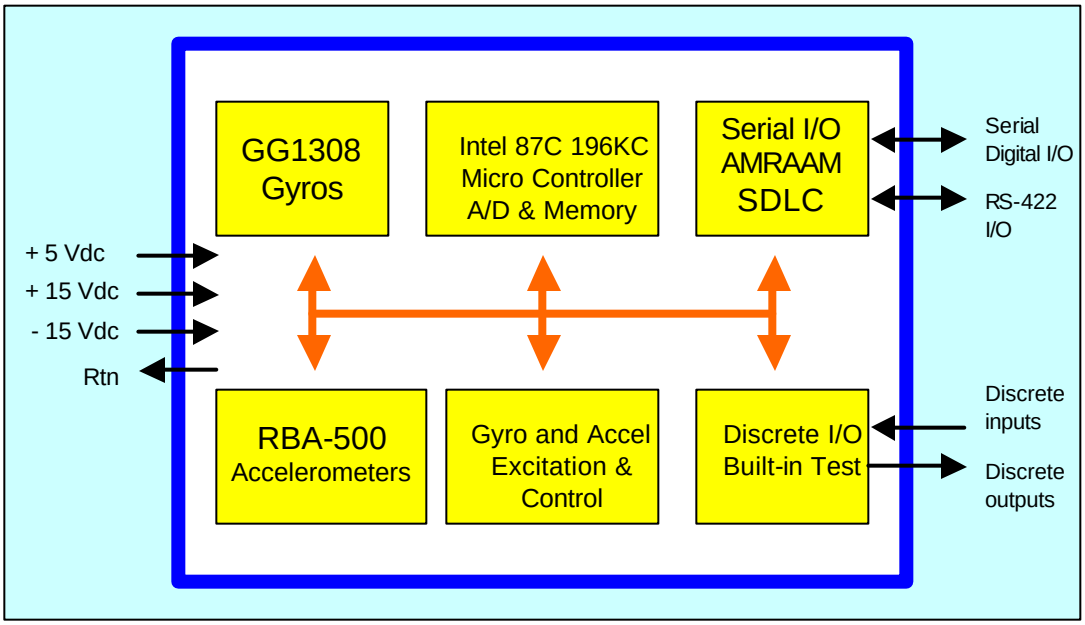


Figure 5.1: IMU Block Diagram [Hon00].

Table 5.2: IMU System Specifications.

Parameter	HG1700AG11
Dimension (in)	3.7 ϕ x 2.9 H
Volume	< 33 in^3
Weight	< 2.0 lbs.
Power	< 8 Watts
Bit Effectiveness	> 92%
Life: operating	> 10,000 hrs.
Life: dormancy	> 10 years
Output Data Rate	100 Hz / 600 Hz
ACCELEROMETER CHANNEL	
Operating Range, ± g's	37
Scale Factor, ft/sec	7.4506E-09
Scale Factor Accuracy, ppm 1 σ	300
Scale Factor Linearity, ppm 1 σ	500
Bias, milli-g's 1 σ	1
VRE, micro-g's max	500
Axis Alignment Stability, μrad 1 σ	500
Axis Alignment Stability, (non-orthogonality) μrad 1 σ	100
Output Noise, ft/sec (1 σ/10,000 samples)	0.008
Velocity Random Walk, ft/sec/rt-hr max	0.065
GYRO CHANNEL	
Operating Range, ± g's	1074
Scale Factor, radians	1.1642E-10
Scale Factor Accuracy, ppm 1 σ	150
Scale Factor Linearity, ppm 1 σ to ± 800°/sec	150
Bias, °/hr 1 σ (Drift)	1
Axis Alignment Stability, μrad 1 σ	500
Axis Alignment Stability, (non-orthogonality) μrad 1 σ	100
Output Noise, μrad (1 σ/10,000 samples)	80
Velocity Random Walk, °/rt-hr max	0.125

Source: [Hon00 & Hon99]

IMU Power Requirements

The connections for the power and signal interface of the Honeywell HG1700 AG11 Inertial Measurement Unit are made through a 30 filter pin header. A flexible circuit assembly interconnect hooks directly to the IMU at one end and a 25-pin D-sub at the other end. This flexible cable (see Figure 5.2) minimizes the effective force loading on the IMU and providing sufficient cable relief allows normal motion of the suspended mass (inertial sensor assembly). Table 5.3 details the pin assignments for the IMU (Hon99).

Table 5.3: IMU Header Pin Assignments.

Description	IMU Pin	37P Dsub
+ 5 VDC	10	36 / 18
+15 VDC	1	1 / 20
- 15 VDC	21	35 / 17
GND	11	2 / 21
+ 5 VDC RETURN	9	19 / 37
+15 VDC RETURN	22	3 / 22
+ DATA (Hi)	4	26
- DATA (Lo)	24	25
+ Clock (Hi)	28	31
- Clock (Lo)	6	29

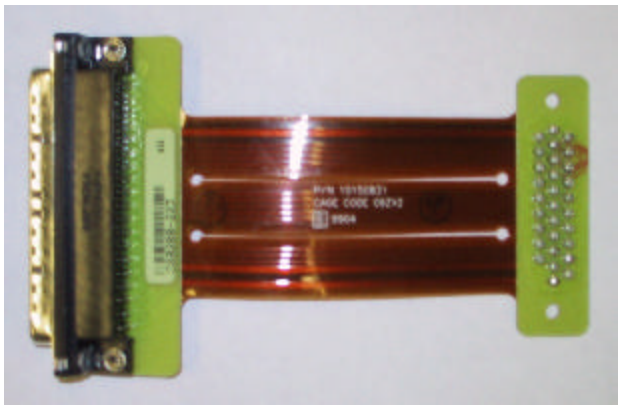


Figure 5.2: IMU Flexible Circuit Assembly Interconnect.

The HG1700AG11 IMU requires input power of +5V and ±15V. The details are listed in Table 5.4. Power is supplied to the IMU through a DC-DC power converter that takes in 12V from the system shelf and outputs +5V and ±15V.

Table 5.4: IMU Power Requirements.

			Starting			Running		
Voltage (VDC)	Tolerance	Max Ripple (mV pk-pk)	Max Current (Amp)	Max Starting Duration (msec)	Max Starting Duration (msec)	Max Current (Amp)	Nominal Current (Amp)	Min Current (Amp)
+ 5	± 5 %	80	0.80	40	25	0.65	0.35	0.252
+ 15	± 5 %	100	0.70	500	60	0.35	0.25	0.16
- 15	± 5 %	100	0.09	20	10	0.08	0.07	0.005

Sensor Axes and Mounting

The Honeywell HG1700AG11 is designed primarily for use in guided missiles, hence its cylindrical shape. Figure 5.3 shows how the sensor axes (X, Y, Z) are oriented.

In order to fix the IMU sensor axes by aligning them with the vehicle axes, an aluminum mount is designed. This mount uses dowel pins precisely located to align the axes in the following manner:

IMU sensor X-axis ←→ Vehicle X-axis (forward)

IMU sensor Y-axis ←→ Vehicle Y-axis (right)

IMU sensor Z-axis ←→ Vehicle Z-axis (down)

The mount is then fixed on the system shelf, which fits into the NTV rear cabinet. Also, the flexible cable interconnect is affixed on the right side of the mount. Figure 5.4 shows the mounted IMU set-up.

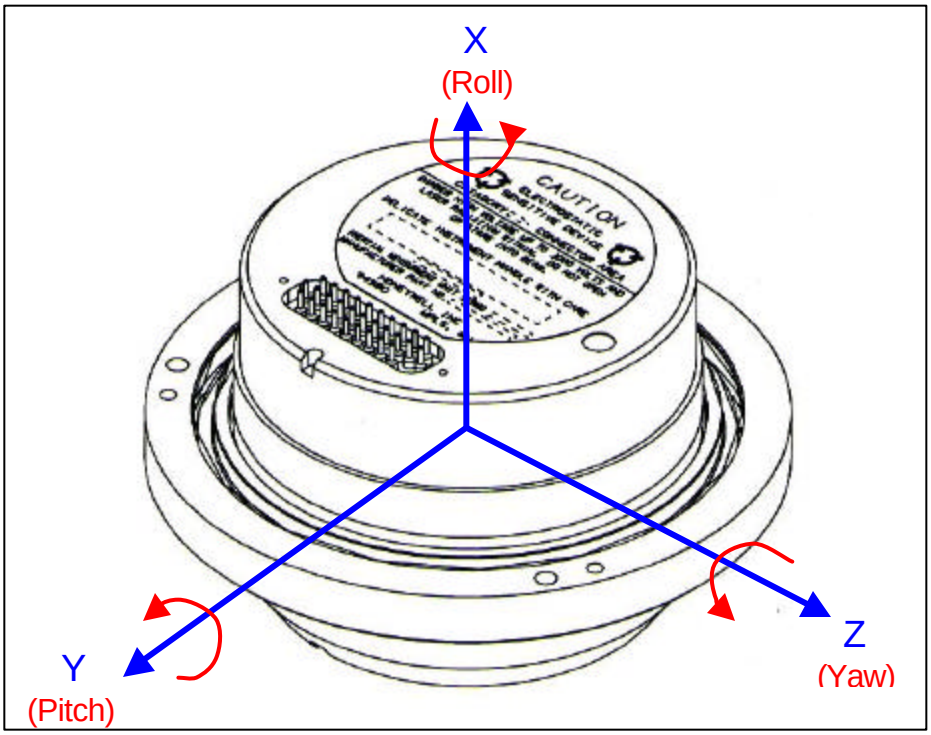


Figure 5.3: IMU Sensor Axes.

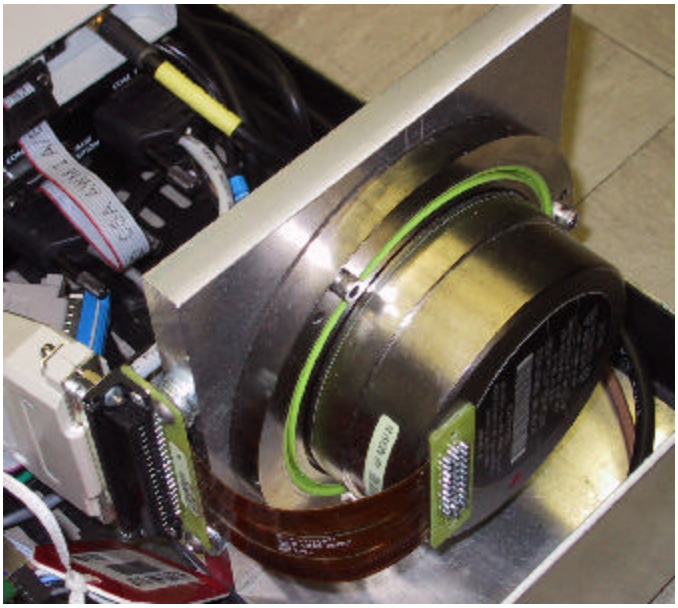


Figure 5.4: Mounted IMU.

Serial Communications Interface

The IMU provides an electrical serial data interface composed of four output and three input differential signal pairs. It employs a SDLC (Synchronous Data Link Control) point-to-point type protocol through an RS422 serial interface.

SDLC Type Protocol Messages

Each SDLC frame opens with a flag followed by the information that can include, address, control and data fields that depend on the SDL protocol implementation used. Following the information are two bytes that report the result of the Cyclic Redundant Check (CRC). The CCITT polynomial used for the CRC is $X^{16} + X^{12} + X^5 + 1$, preset to ones. A closing flag follows the CRC and flags are transmitted continuously between frames. The SDLC message format is described in Figure 5.5. Furthermore, Table 5.5 shows the SDLC Bus characteristics (Hon99).

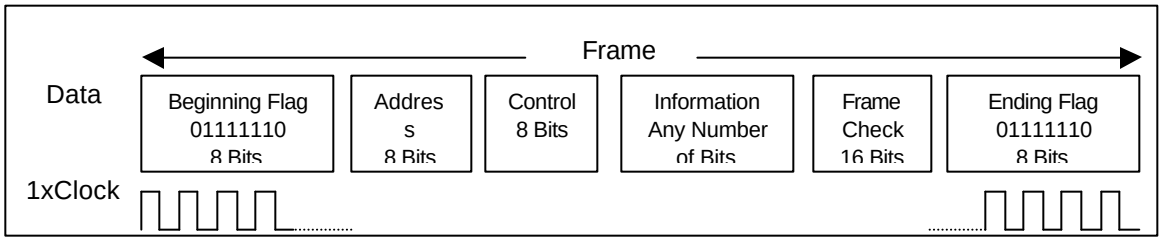


Figure 5.5: SDLC Message Format.

The IMU starts transmitting the data once the IMU host computer supplies the receive clock to the IMU and the IMU performs the initialization sequence. The initialization sequence shall be less than 400 milliseconds from power application.

Table 5.5: SDLC Bus Characteristics.

Parameter	Description			
Shift Clock	1.0 MHz provided by the host (GCU)			
Data Encoding	NRZ (non-Return to Zero)			
SDLC Mode	Point-to_point			
CRC	CCITT			
Data Ordering	Least Significant (LS) bit first; LS byte first; LS 16 bit word first			
Message Format	Message from GCU to IMU	Message from IMU to GCU		
	NONE USED	Description	Bytes	Value/Para
		Flag	1	7Eh
		IMU Addr	1	0Ah
		IMU Msg ID	1	Table 5.6
		Data	n	Table 5.7, 5.8
		CRC	2	16-bit poly
		Flag	1	7Eh

The IMU data messages are divided into 600Hz flight control data and 100Hz inertial data. Table 5.6 gives the IDs for both message types. The start of the sequential transmissions of the message ID number 1 are on the average 1/600 ± 0.01% seconds apart. The start of the sequential transmissions of the message ID number 2 are on the average 1/100 ± 0.01% seconds apart.

Table 5.6: IMU Data Message ID.

Transmitted Messages			
IMU Msg ID	Title	Source	Destination
1	Flight Control Data	IMU	System Processor
2	**Flight Control and Inertial Data**	**IMU**	**System Processor**

Tables 5.7 and 5.8 describe both flight control and inertial data messages. For the IMU to begin transmitting output data, the IMU receive clock should be set to 1.0 MHz ± 2% by the host computer through the RCV_CLK differential input pair. The RCV_CLK specifications

are listed in Table 5.9. Data is transmitted through the SER_DATA differential input pair. The Transmit Data lines are described in Table 5.10. All the tables are taken from the Honeywell HG1700AG11 Reference Manual.

Table 5.7: Flight Control Data.

Item	Parameter	# Bytes	LSB Value	Units
1	Angular Rate Body X	2	$2^{-20}*600$	rad/sec
2	Angular Rate Body Y	2	$2^{-20}*600$	rad/sec
3	Angular Rate Body Z	2	$2^{-20}*600$	rad/sec
4	Linear Acceleration Body X	2	$2^{-14}*600$	ft/sec^2
5	Linear Acceleration Body Y	2	$2^{-14}*600$	ft/sec^2
6	Linear Acceleration Body Z	2	$2^{-14}*600$	ft/sec^2
7	Status Word 1	2	-	-
8	Status Word 2	2	-	-

Table 5.8: Flight Control Data and Inertial Data.

Item	Parameter	# Bytes	LSB Value	Units
1-8	Flight control Data	16	-	-
9	Delta angle - body X	4	2^{-33}	radians
10	Delta angle - body Y	4	2^{-33}	radians
11	Delta angle - body Z	4	2^{-33}	radians
12	Delta velocity - body X	4	2^{-27}	ft/sec
13	Delta velocity - body Y	4	2^{-27}	ft/sec
14	Delta velocity - body Z	4	2^{-27}	ft/sec

Table 5.9: Receive Clock (RCV_CLK) Line Specifications.

Parameter	Characteristic
Signal Type	Differential digital input; RS-422 compatible
Differential Input Threshold Voltage	0.2 volts minimum
Input Resistance	180 ohms ± 5% due to 180 ohm termination resistor between input pair
Period	1.0 µsec ± 2%
Pulse Width	0.5 µsec ± 2%
Rise Time	IAW RS-422
Fall TIme	IAW RS-422
Ground Resistance	5 V RTN

Table 5.10: Transmit Data (SER_DATA) Line Specifications.

Parameter	Characteristic
Signal Type	Differential digital input; RS-422 compatible
High-Level Output Voltage (V_{OH})	2.5 volts minimum; I_{OH} = -20 mA
Low-Level Output Voltage (V_{OL})	0.2 volts maximum; I_{OL} = 20 mA
Load Impedance	180 ohms ± 5% due to 180 ohm termination resistor between input pair
Period	1.0 μsec ± 2%
Pulse Width	1.0 μsec/bit
Rise Time	IAW RS-422
Fall TIme	IAW RS-422
Ground Resistance	5 V RTN

The latency between the time the sensor data is read from the internal IMU registers to the start of transmission of the flight control and inertial message shall be 1.693 ± 0.005 milliseconds.

Sealevel ACB104 Serial Communications Controller Card

The serial communications controller card selected, the ACB104, has one high-speed, RS-232/530/422/485 synchronous/asynchronous port capable of processing SDLC protocol messages. The ACB-104 (see Figure 5.6) is PC-104 compatible and is the same serial card used in interfacing with the H-726 MAPS. To accommodate the 1.0 MHz data sync speed, the ACB104 will be configured to use Direct Memory Access, or DMA. DMA is a method of data transfer that allows information to be transferred from memory to an I/O device, bypassing the CPU, thus making it fast and efficient. Maximum data rates obtainable using DMA reach 1.5 to 2.0 MHz (Mbps). DMA also frees up the CPU to handle concurrent calculations and processes. A DMA driver written specifically for the ACB104 which runs under DOS is used. When run, the DMA driver loads a driver configuration file that includes all the driver specific

settings. These settings are listed in Table 5.11. Additionally, the hardware jumper settings for DMA mode are enumerated in Table 5.12.

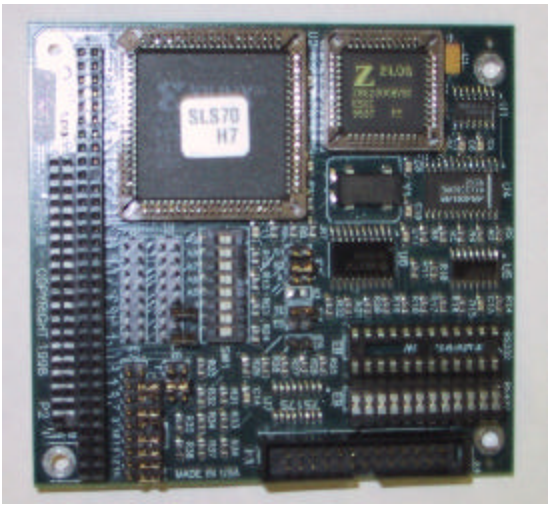


Figure 5.6: Sealevel ACB-104.

Table 5.11: SeaMAC DMA Driver Configuration.

Parameter	Setting	Comments
IRQ	5	Interrupt Channel
BaseIO	3E8	Base Address
TxDMA	3	Transmit Channel
RxDMA	1	Receive Channel
TxBufferSize	1024	Max Size of Transmit Frame in Bytes
RxBufferSize	1024	Max Size of Receive Frame in Bytes
TxBuffers	2	No. of Transmit Frames in queue
RxBuffers	30	No. of Receive Frames in queue
TxClock	BRG	Transmit clock set to Baud Rate Generator
RxClock	BRG	Receive clock set to Baud Rate Generator
Divisor	0x00	Clock Divisor
DPLL	NRZ	Non Return to Zero
SWInt	65	Software Interrupt
NodeAddress	None	0 - 0xff or None

Table 5.12: ACB-104 DMA Jumper Settings.

Jumper Header	Function	DMA Setting
E1	DMA Enable	A
E2	DMA Channel (half-duplex)	11
E3	DMA Channel (full-duplex)	None
E4	RS-485 Mode	None
E5	Transmit Clock	None
E6	IRQ Mode	T
E7	IRQ Selection	5

IMU Data Processor (IDP)

The IMU Data Processor is in charge of reading the raw data, converting it to usable values, checking for bad data, averaging the data, and outputting it to the primary computer (POS). The Data Processor (DP) is a PC-104 stack (Pentium 133 MHz, see Figure 5.5) with a 640 MB Calluna hard drive. As described earlier, the IMU is interfaced to the DP through an ACB-104 serial communications card. The data flow in the IDP is shown in Figure 5.7.

Raw Data Conversion

Upon establishing a solid data link between the DP and IMU, the next step is to process the incoming data. First, the inertial data messages are selected from the data stream by checking the message type byte. Coming in at 100 Hz, the inertial data message is then parsed and the actual data values are calculated by multiplying the raw data with the LSB values described in Table 5.8. The method for bit conversion is as follows:

$$\textit{Datafield Value} = \textit{base10(Byte1)} + \textit{(base10(Byte2))}16^{2} + \textit{(base10(Byte3))}\,16^{4} + \textit{(base10(Byte3))}16^{6} \qquad [5.1]$$

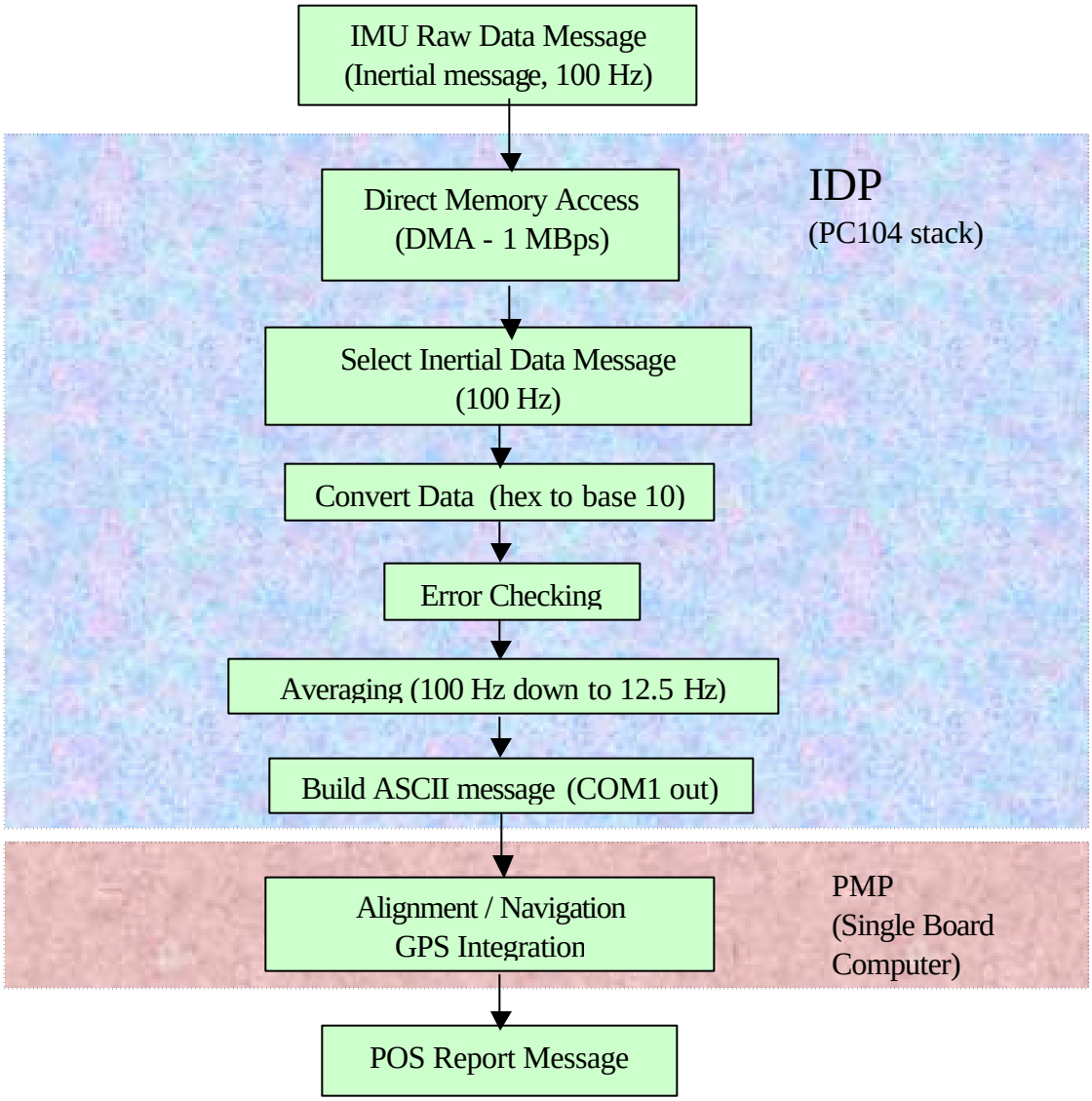


Figure 5.7: IMU Data Processor (IDP) Data Flow.

The output of each of the six IMU data fields are represented in Table 5.13. Their corresponding value ranges are shown in Table 5.14.

Table 5.13: IMU Data Field Representation.

	IMU Data Field Values (4 bytes)
Maximum Value (hex)	ffffffff
Maximum Value (b10)	4,294,967,295
Minimum Value (b10)	0
Positive Range	Min Val ←→ ((Max Val – 1) / 2)
Negative Range	((Max Val + 1) / 2) ←→ Max Val
b10 – base 10 ; hex - hexadecimal	

Table 5.14: IMU Data Field Value Range.

Datafield [over 0.01 sec]	Minimum Value	Maximum Value
Delta Theta (X, Y, Z) [rad]	- 0.25 [-14.32 deg]	+ 0.25 [+14.32 deg]
Delta Velocity (X, Y, Z) [ft/sec]	- 16.0 [-50 g]	+ 16.0 [+50 g]
* Conversion : Delta Theta = (base 10 value) * 2^{-33} Delta Velocity = (base 10 value) * 2^{-27}		

Error Checking

The next step is to perform error checking. This prevents unwanted occurrences of bad or corrupted data. Acceptable data ranges are set by minimum and maximum threshold values for both delta theta and delta velocity (see Table 5.15). Since under normal conditions, the IMU will be subjected to accelerations and torques significantly below their maximum measurable limits, the threshold values are manually determined to match extreme operating conditions based on the type of intended application. Data that exceed the threshold values will be thrown out and replaced by data from the previous time step.

Table 5.15: IMU Data Field Limit and Threshold Values.

Data Field	IMU Limit	Threshold Value	TV / period
Δ Theta (rad)	± 0.1745	± 0.1	± 10 rad/sec
Δ Theta Dot (rad)	-	± 0.04	± 4 rad/sec
Δ Velocity (ft/sec)	± 16.1	± 8	800 ft/sec^2
Δ Velocity Dot (ft/sec)	-	± 1.0	100 ft/sec^2

Data Averaging

To reduce the computational burden of the navigation processor, the data is averaged to a rate fast enough to satisfy autonomous navigation requirements yet slow enough not to compromise the computational efficiency of the integrating computer. Another factor is the allowable output data rate from the data processor. Using an RS-232 serial output line (at 19200 baud), IMU messages are output up to 12.5 Hz. Comparatively, the MAPS currently outputs messages at 12.5 Hz.

The method of averaging involves calculating the total value for each data field for every 8 data points. These total values are then divided by the average data period (1/12.5 Hz = 0.08 sec) to get average values for omega (angular velocity) and acceleration. Acceleration is converted from feet/sec^2 to m/sec^2.

IMU Output Message

To make the IMU data usable for navigational computation, the data is converted to angular rates and linear accelerations. As pointed out earlier, the IMU outputs such data in standard message sets at a rate of 12.5 Hz. The format of the IMU message output from the data processor is given in Table 5.16.

Table 5.16: IMU Output Data Message Format.

Field	Description	Value
1	Header	$IMU
2	Omega X (rad/sec)	12.345678
3	Omega Y (rad/sec)	12.345678
4	Omega Z (rad/sec)	12.345678
5	Acceleration X (m/sec^2)	12.345678
6	Acceleration Y (m/sec^2)	12.345678
7	Acceleration Z (m/sec^2)	12.345678

Sample Message: $IMU,-4.234926,0.002433,0.526343,1.434125,0.000043,9.784353

The IMU output data message is sent out through the COM1 serial port of the PC104 stack. The serial connection is set at 19200 baud and is fed into the COM2 serial port of the Main POS computer (SBC).

The software code running on the IMU Data Processor stack is shown in Appendix B.

Novatel RT-20 GPS

Novatel's RT-20, real-time kinematic Differential GPS (DGPS) receiver, delivers one meter performance at power-up. After a brief period (as less as 3 minutes), robust and reliable double differencing techniques provide real-time accuracies of up to 20 centimeters or better.

The primary advantage of the RT-20 (see Figure 5.8) is that it fills the accuracy gap between pseudorange positioning and fixed ambiguity carrier phase positioning. The RT-20 has been used extensively by CIMAR and AFRL, employing it in systems such as the ordnance-detecting ASV.

Figure 5.8: Novatel PowerPak RT-20 GPS receiver.

System Specifications

The specifications for the Novatel PowerPak RT-20 are given in Table 5.17.

Table 5.17: Novatel PowerPak RT-20 Specifications.

Parameter	Specification
Position Accuracy, standalone SA on	40 m CEP
Position Accuracy, standalone SA off	15 m CEP
Position Accuracy, differential RTCM	0.75 m CEP
Position Accuracy, differential RT-20	0.20 m CEP
post-processed	± 5 mm + 1 ppm
Time to First Fix, cold start	67 sec (typical)
Re-acquisition, warm start	1 sec (typical)
Data rate, Raw measurements	20 Hz
Data rate, position	10 Hz
Data rate, RT-20	5 Hz
Time accuracy, SA on	250 ns RMS
Time accuracy, SA off	50 ns RMS
Velocity accuracy, standalone	0.20 m/s RMS
Velocity accuracy, differential	0.03 m/s RMS
Measurement precision: C/A code	10 cm RMS
L2 P code	40 cm RMS
L1 carrier phase single channel	3 mm RMS
L1 carrier phase differential channel	0.75 mm RMS
L2 carrier phase single channel	5 mm RMS
L2 carrier phase differential channel	4 mm RMS
Dynamics, acceleration	6 g
Dynamics, velocity	515 m/s
Physical Measurements	20.8 cm x 11.1 cm x 4.7 cm
Weight	1 kg
Temperature, operating	-40 °C to +65 °C
Temperature, storage	-40 °C to +85 °C
Humidity	≤ 95% non-condensing
Communications Interface	RS232/RS422/NMEA
Baud Rate	300 to 115,200 bps
strobe I/O	5 signals, TTL level
Communications connector	2 x DB9P
Strobe I/O connector	DB9S
Antenna connector	TNC female
Power connector	2.1 mm threaded plug (center +)
Input voltage range	10-36 VDC
Power consumption	8 watts (typical)

Figures 5.9 and 5.10 illustrate typical performance curves for position accuracy convergence for both static and kinematic modes. It must be noted that the Novatel ProPak Beeline described in Chapter 4 uses the same RT-20 millenium GPS card and thus shares the same performance specifications. Also, in testing the beeline, the static and kinematic convergence statistics are verified. As explained in the previous chapters, CEP refers to Circular Error Probable which represents an easting-northing circle where 95% of all 2D position errors fall into.

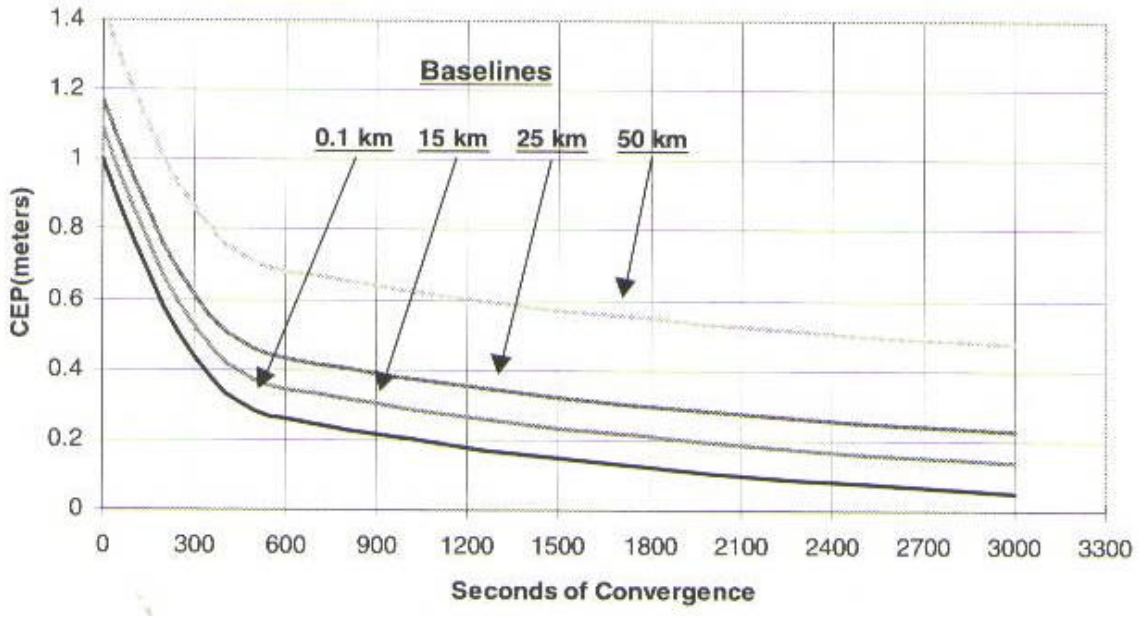


Figure 5.9: Typical RT-20 Convergence - Static Mode.

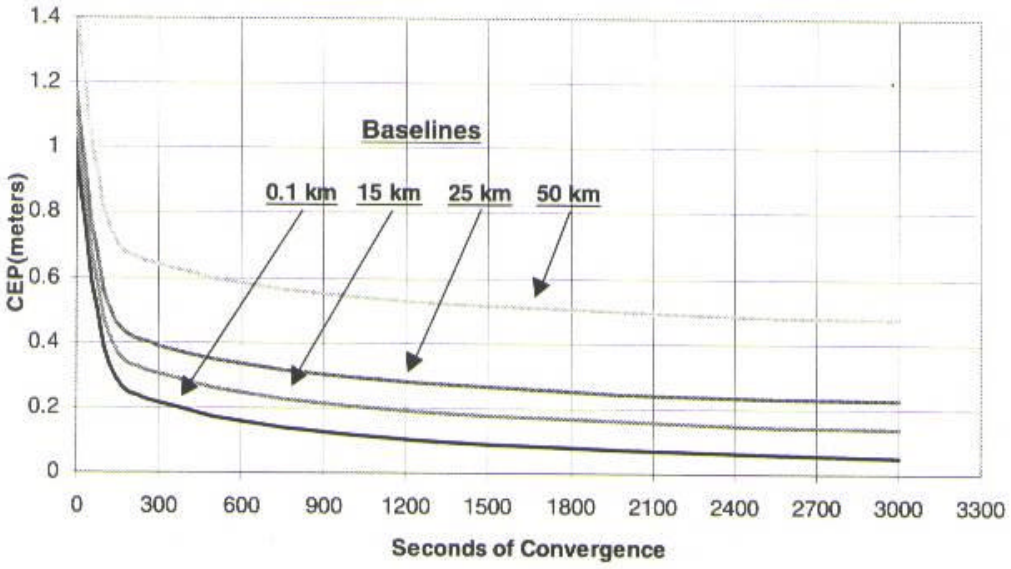


Figure 5.10: Typical RT-20 Convergence - Kinematic Mode.

System Features

(1) 20-cm real-time kinematic (RTK) accuracy with "on-the-fly" initialization

(2) L1 C/A code and carrier phase tracking

(3) 12 channel "all in view" parallel tracking

(4) Fast reacquisition

(5) Patented Narrow Correlator tracking technology

(6) Multipath Elimination Technology (MET)

(7) 1 PPS output

(8) Event marker

(9) RTCM SC104 v 2.1/2.2, RTCA SC159

(10) GPSolution - Windows compatible graphical user interface (GUI)

The DGPS set-up includes two RT-20 receivers each connected to a Novatel 501 GPS antenna w/ choke ring and powered by a 12 VDC source. The rover receiver (mounted on the vehicle) receives differential corrections from the base receiver through RF. The RF link is established with two Breezecom wireless ethernet radios. The distance between the rover and the base receivers is called the baseline. To assure solid data link and better GPS performance, a baseline of less than 200 meters is maintained at all times. A typical DGPS set-up is depicted in Figure 5.11.

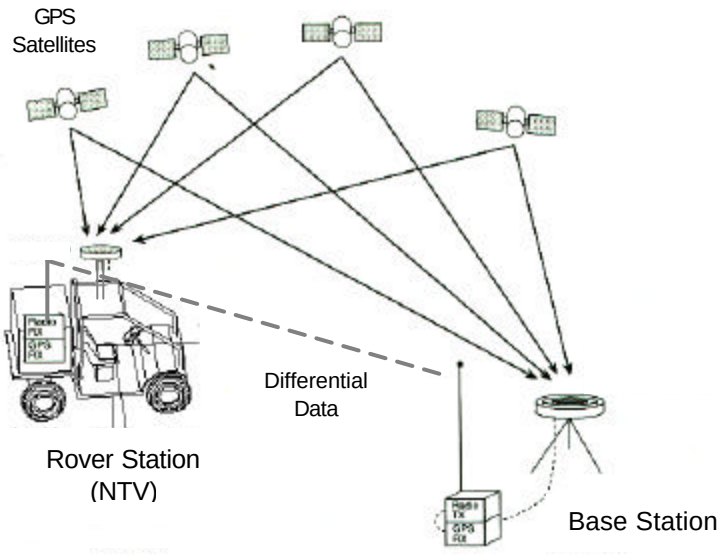


Figure 5.11: Differential Global Positioning System (DGPS) Set-up.

Message Sets

For the purpose of IMU integration, GPS real-time kinematic position data is needed. The essential RT-20 output data is contained in the PRTKA/B log. This can either be read in

as an ASCII-type or binary message. For ease of data processing, logs will be read in ASCII (PRTKA).

The PRTKA log contains the best available position computed by the receiver, along with three status flags. It reports other status indicators, including a differential lag, which is useful in predicting anomalous behavior brought about by outages in differential corrections. Table 5.18 enumerates the individual fields in the PRTKA log [Nov97].

Table 5.18: PRTKA Log.

Field #	Field Type	Data Description	Example
1	$PRTKA	Log header	$PRTKA
2	week	GPS week number	872
3	sec	GPS time into the week (in seconds)	174963.00
4	lag	Differential lag in seconds	1.000
5	#sv	Number of matched satellites	8
6	#high	No. of matched sats above RTK mask angle	7
7	L1L2 #high	No. of matched sats above RTK with L1L2	7
8	latitude	Latitude of position in deg (+ North)	51.11358042429
9	longitude	Longitude of position in deg (+ East)	-114.043580067
10	height	Height of position in meters above sea level	1059.4105
11	undulation	Geoidal separation in meters	-16.2617
12	datum ID	Current datum	61
13	lat σ	Standard deviation of latitude in meters	0.0096
14	lon σ	Standard deviation of longitude in meters	0.0100
15	height σ	Standard deviation of height in meters	0.0112
16	soln status	Solution status	0
17	rtk status	RTK status	0
18	posn type	Position type	4
19	idle	Percent idle time (%)	42
20	stn ID	Reference station identification	119
21	*xx	Checksum	*51
22	[CR][LF]	Sentence terminator	[CR][LF]

Sample Message:
$PRTKA,872,174963.00,1.000,8,7,7,51.11358042429,-114.043558006710,1059.4105,-16.2617,61,0.0096,0.0100,0.0112,0,0,4,42,119*51[CR][LF]

System Integration

Once the inertial and GPS components have been selected and their data interfaces have been specified, system integration follows. The integration becomes an essential step since in a very real sense the INS and the GPS are now mutually aiding, each covering the other's shortcomings and benefiting the overall system performance. The INS keeps position, velocity and attitude (PVA) current, while the GPS provides external measurements to the Kalman filter (KF) thus producing the most probable position (MPP) in the overall navigation solution.

There are basically two approaches to system integration, "loosely-coupled" and "tightly-coupled". In a loosely integrated GPS/INS system, GPS position, rather than raw output, is input into the navigation computer KF. Because the GPS receiver already has a KF of its own, the system has a filter-driving-filter arrangement. This is susceptible to stability problems when the KF gains were changed, or "tuned" to boost performance. These gains are considered "loose", that is, they are left alone. The system's merit is that the GPS could be operated in a stand-alone mode.

The INS/GPS system has evolved into a "tightly coupled" architecture where the navigation processor KF accepts raw inputs from both the GPS and IMU. There is no KF in the GPS processor. Unmodeled errors from the GPS KF are eliminated, and this allows the KF gains to be tuned within smaller limits [Bie99].

Since the Novatel RT-20 GPS receiver already possesses its own KF, the IMU/GPS system is "loosely-coupled". This also lends to the fact that one of the main motivations of the development of this system is to replace the existing Honeywell MAPS. The advantage of a "loosely-coupled" system is it allows flexibility in

interchanging the components. When less expensive, higher accuracy components become available, system integration can be done with minimal change to the system design. Table 5.19 lists the advantages and disadvantages of a loosely-coupled as compared to a tightly-coupled system.

Table 5.19: Loosely-coupled Integration.

Advantages	Disadvantages
1. Lower level modularity – easily change KF to model different IMUs (with different output data variables). Primary KF tailored to IMU type. 2. Able to directly compare MAPS with IMU + primary KF using the existing POS external KF set-up. Cross comparison by switching components. 3. Faster development – since integration with GPS is already done by the secondary KF. Work needed only to develop primary KF for IMU.	1. Less efficient – since primary KF into secondary KF - redundant operations in position, orientation and velocity conversion and error determination. Increased software complexity. 2. Need more processing power – primary KF processes running at 100 Hz and secondary KF running at 10 Hz. Need separate CPUs (added problem of communication between CPUs). 3. Higher cost stemming from additional hardware.

System Configuration

Once the integration approach has been identified, the next step is to apply it in both hardware and software. The general system configuration is shown in Figure 5.12.

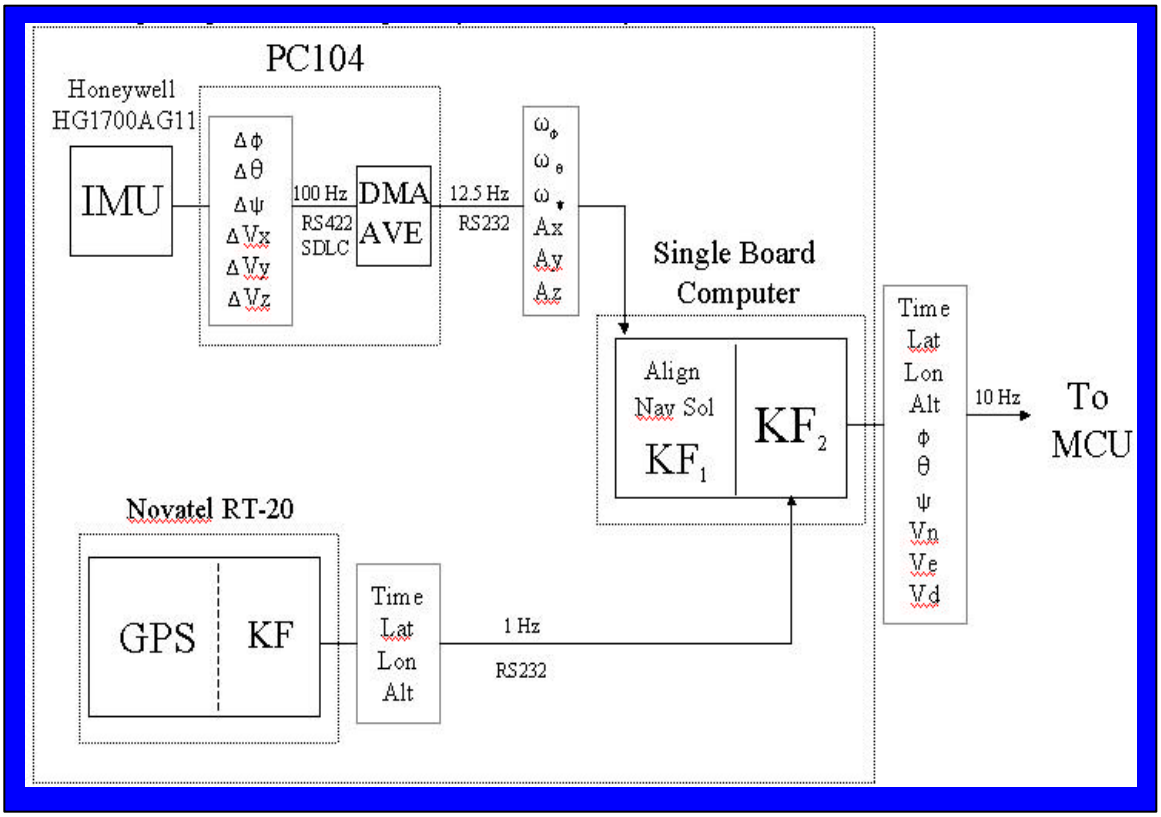


Figure 5.12: IMU/GPS System Configuration.

Hardware

As explained earlier, the IMU is interfaced to an IMU data processor in the form of a PC104 stack running under MS-DOS. To handle system integration, a second CPU is used. This is explained in more detail in the succeeding section on 'POS Main Processor'.

The remaining hardware requirements include individual DC-DC converters for the IMU/IMU Data Processor (DP) and SBC and a Freewave Wireless Transceiver (radio modem) for reception of DGPS base corrections. The input voltage to the system shelf is 12 VDC. Main power is first routed to the SBC, radio modem and GPS receiver. A secondary switch turns on the IMU and the IMU DP. This enables the user to debug the system independently as well as reset the IMU without rebooting the main POS computer

(SBC). Figures 5.13 and 5.14 illustrate the electrical diagram and hardware layout of the POS system shelf. Table 5.20 provides a breakdown of the power requirements.

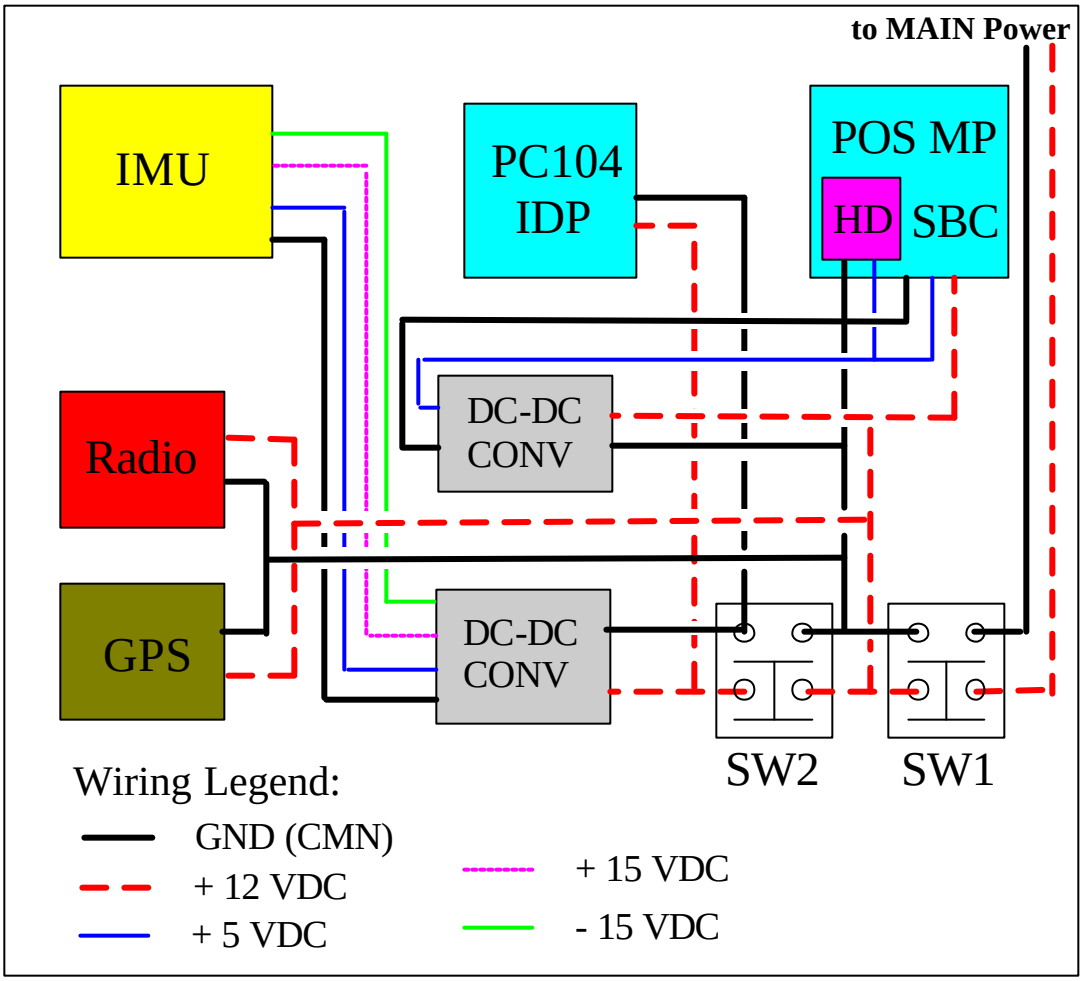


Figure 5.13: POS System Shelf Electrical Diagram.

Table 5.20: POS System Shelf Power Requirements.

System Component	Voltage (VDC)	Current (Amp)	Power (W)
SBC (POS MP) - CPU	+ 5 V	3.5	18
- HD	+ 5 V	0.7	3.5
- PC104 Ethernet card	+ 12 V	0.5	6.0
PC104 (IMU DP)	+ 12 V	0.6	7.2
	+ 15 V	0.25	3.75
IMU	- 15V	0.07	1
	+ 5V	0.35	1.75
Novatel PowerPak RT-20 GPS	+ 12 V	0.65	8.0
Freewave Data Transceiver	+ 12 V	0.5	6.0
POS System Total	+ 12 V	5.0	60.0

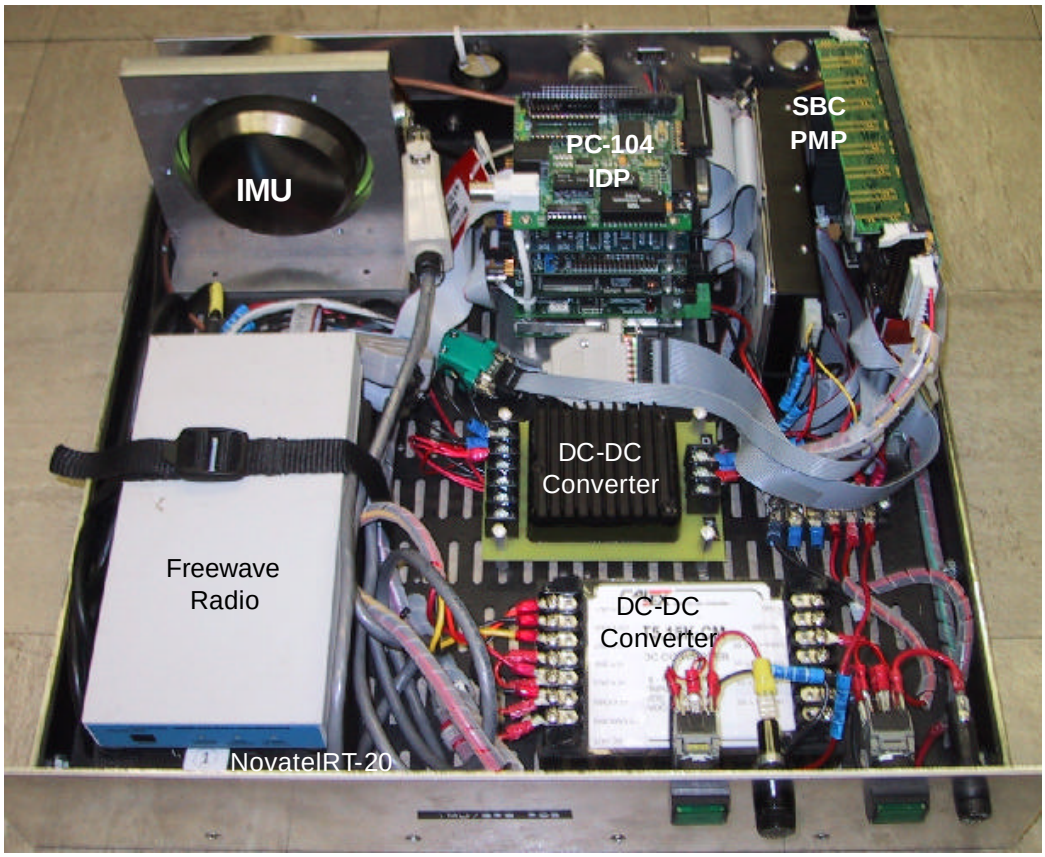


Figure 5.14: POS System Shelf Hardware Layout.

Software

The SBC is configured to run under the Red Hat Linux 7.1 Operating System. The software design uses the same standard approach in defining position process source code and library files. The main differences from the previous approach similar to the MAPS/Ashtech POS system include:

1. Component messages are JAUGS compatible as defined by the latest document version. Position system has component ID 38 [Joi00].
2. Instead of position system messages being output directly to the Mobility Control Unit (MCU), a Message Routing System (MRS) allows higher level system components to access position data without interrupting other systems.
3. The position system processes are lumped together in a single POS directory where the main program, pos.c, spawns off the individual process threads. These processes call functions contained in process libraries described below:

pos : This is the main program which takes charge of starting and monitoring all the processes threads performed by the IMU, GPS and Filter programs. All the component messages are also handled here.

ImuLib : This handles all communications between IMU Data Processor (PC104) and the POS main computer. This includes receiving and processing of IMU raw acceleration and angular velocity data, receiving of GPS position updates and system commands. Also, initial alignment and navigation solution algorithms process IMU data for sending out to the Kalman filter.

gpsLib : This block reads in GPS position messages from the Novatel RT-20 and stores the data in global variables. Also, a 1 PPS (Pulse Per Second) signal is used to reset the SBC system clock every minute, thereby synchronizing it with GPS time.

filterLib : This part starts off the external Kalman filter individual processes in sequence while receiving inputs from the IMU (navigation solution) and GPS. The output position and orientation data is then translated to the vehicle control point.

posConsole : This program is outputs data on screen. It shows Filter, GPS, and IMU position and orientation data, as well as system and component status.

mrs : This handles all data transferred to and from the POS and the network. This is run in the background separately prior to start of the pos main program.

The software structure is graphically represented in Figure 5.15.

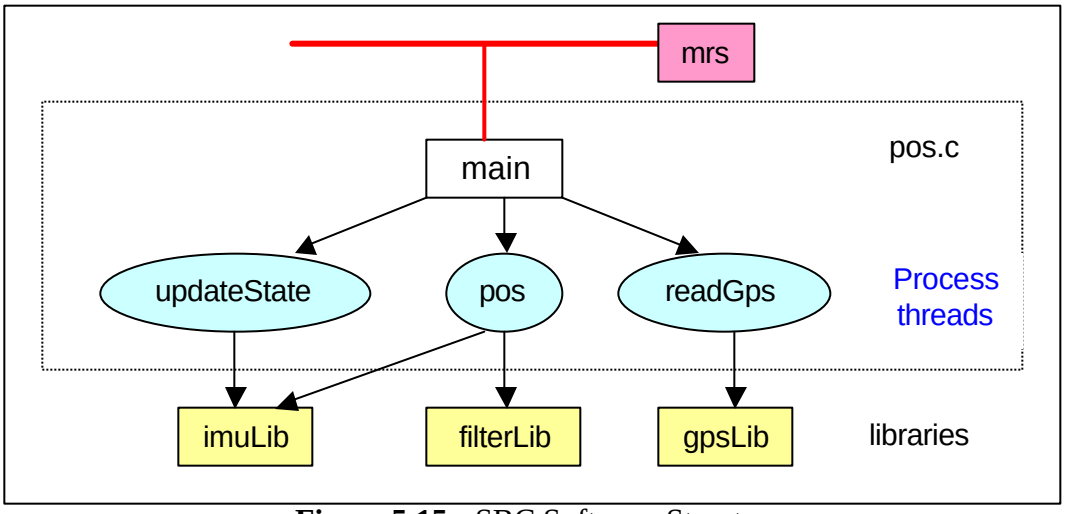


Figure 5.15: SBC Software Structure.

POS Main Processor (PMP)

The POS Main Processor is in charge of integration of the IMU and GPS. A Winsystems EBC-TX Plus Single Board Computer (SBC) with a 333 MHz AMD processor, 128Mb of RAM, and a 30 GB hard drive provides it more computing power to handle multiple tasks of running the navigation solution, Kalman filter, and data message input/output. The system specifications of the SBC are given in Table 5.21. In addition, Figure 5.16 shows the data and process flow through the PMP.

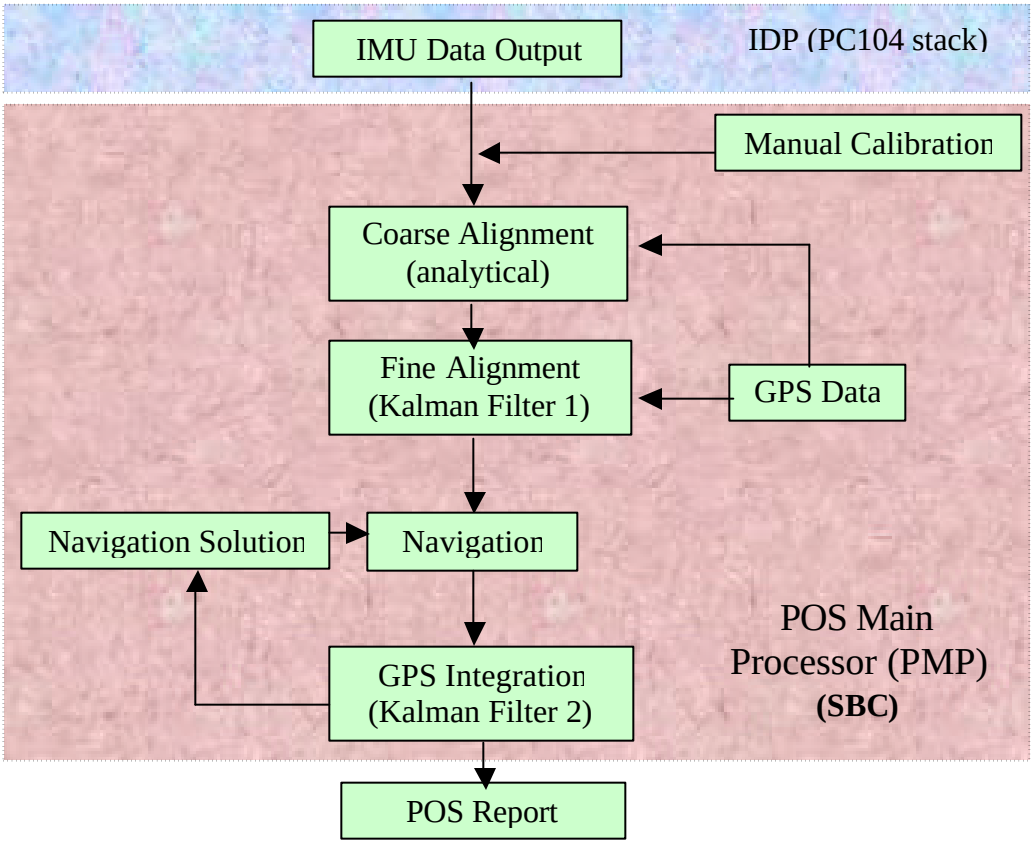


Figure 5.16: POS Main Processor (PMP) Data and Process Flow.

Table 5.21: Winsystems EBC-TX Plus SBC.

Parameter	Specification
CPU	333 MHz AMD Triathlon
RAM	128 MB SDRAM
Hard Drive	Sandisk 350MB Flashdisk
Features	onboard Ethernet, digital I/O, video card
Ports	4 COM ports, 1 parallel port
PC104 Expansion Slot	RTD CMT104 HD carrier module
Input Voltage	+5 VDC, +12VDC (for PC104 expansion slot)
Input Current	3.5 A (CPU), 0.7 A (HD),
Power	25 W
Size	8” x 5” x 2” (w/ HD)

IMU Calibration

To accurately interpret the raw data output from the IMU, a manual calibration procedure should be performed. Eventhough the IMU has already been calibrated at the factory, manual calibration ensures continued accurate performance by measuring current inherent accelerometer biases and gyroscope drifts. These calibration coefficients (biases and drifts) will then be applied to the raw data output to obtain values for delta velocities and delta angles closer to the true values.

This manual calibration procedure should be performed carefully, paying close attention to specific details. It should be noted that the accuracy of the calibration will greatly affect the way the IMU data is used for alignment and eventually the performance of the system. Routine re-calibration should be done to make sure system performance does not degrade over time.

Sources of Error

There are different types of errors associated with ring laser gyros (RLG) and accelerometers. These errors are essentially categorized into errors associated with

sensor linearity, forces affecting sensor operation, a fixed sensor output shift (fixed bias) and random offsets (random bias). The bias and drift specifications of the sensors refer to the fixed bias component. These fixed biases are to be determined during manual calibration. Table 5.22 compares the fixed biases of the HG1700AG11 IMU with the H-726 MAPS.

Table 5.22: Fixed Biases of the HG1700AG11 IMU and H-726 MAPS.

Bias Type	HG1700AG11	H-726 MAPS
Accelerometer Bias	1 mg (0.01 m/sec^2)	0.03 mg (0.0003 m/sec^2)
Gyro Drift	1.0 °/hr (5 x 10^{-6} rad/s)	0.01 °/hr (5 x 10^{-8} rad/s)

It is obvious from above that the quality of the sensors of the MAPS is far superior to those of the IMU. It should be noted that orientation directly affects position. Therefore, the accuracy of the IMU is greatly diminished compared to the MAPS.

It should also be noted that the other sources of error contribute significantly to sensor performance. Of main concern is the random bias errors. For an RLG, the random bias term yields random walk. Random Walk (RW) is the root-mean square of angular output which grows with the square-root of time. While for an accelerometer, random bias is caused by instabilities within the sensor assembly [Tit97].

Random bias becomes critical because of its unpredictability. These are manifested in the sensor output as noise. Smoothing can be done to lessen the effects of noise, however, care must be taken not to smoothen out the dynamics within the signal.

An important characteristic of inertial navigation systems is the presence of Schuler oscillations. Professor Max Schuler defined the Schuler pendulum as a mass suspended by a string with length equal to the radius of the Earth. The string would normally define the direction of the local vertical. However, since the Earth rotates, the

support point is moved from rest with an acceleration a, and the string will be deflected by an angle, $\theta = \tan^{-1}(a/g)$. The effect of the Schuler pendulum causes oscillations in the position and orientation output of the inertial navigation system. The Schuler frequency is calculated as

$$\omega_s = \sqrt{(R_0 / g)} = 0.00124 \text{ rad/s} \tag{5.2}$$

and the Schuler period is

$$T_s = \frac{2\pi}{\omega_s} = 84.4 \text{ min.} \tag{5.3}$$

Therefore, the Schuler oscillations will occur every 84.4 minutes. Schuler tuning improves system performance by accurately indicating the vertical with respect to gravity on a moving vehicle navigating close to the surface of the earth.

Table 5.23 provides a list of the error propagation equations for a single sensor axis as a function of the different error sources. It is apparent that over long periods of time, several Schuler periods or more, the errors in a simple navigation system are bounded as a result of the Schuler tuning. This is true for all sources of error with the exception of the drift of the gyroscope which gives rise to a position error which increases linearly with time, $\delta\omega_{yb} R_0 t$, in addition to an oscillatory component. This makes the gyroscope drift the key error source [Tit97].

Table 5.23: Single Axis Error Propagation.

Error Source	Notation	Position Error
Initial Position error	δx_0	δx_0
Initial Velocity error	δv_0	$\delta v_x \frac{\sin \omega_s t}{\omega_s}$
Initial Attitude error	$\delta \theta_0$	$\delta \theta_0 R_0 (1 - \cos \omega_s t)$
Fixed Accelerometer Bias	δf_{xb}	$\delta f_{xb} \left(\frac{1 - \cos \omega_s t}{\omega_s^2} \right)$
Fixed Gyroscope Drift	$\delta \omega_{yb}$	$\delta \omega_{yb} R_0 \left(t - \frac{\sin \omega_s t}{\omega_s} \right)$

Geodetic Constants

To accurately model the gravity attractions at any point on the Earth, the reference Earth model known as World Geodetic System of 1984 (WGS 84, see Figure 5.17). The cross section of the mean sea level surface of the Earth is called the geoid. The geodetic vertical is everywhere normal to the reference ellipsoid [Cha97].

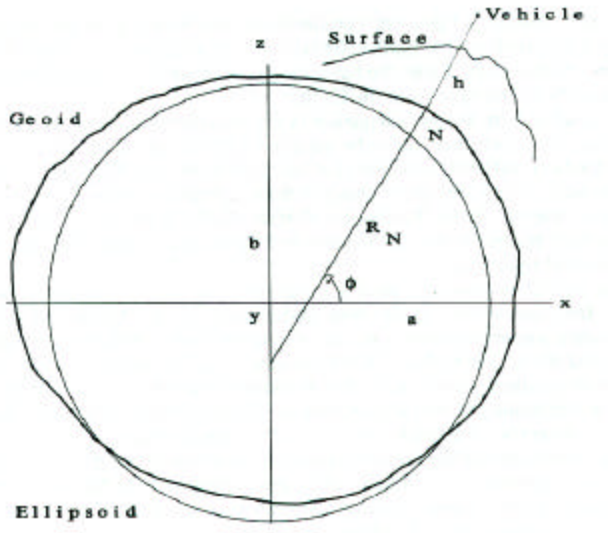


Figure 5.17: Geometry of WGS 84 Reference Earth Model.

The important WGS 84 geodetic constants are given below:

$\omega_{ie}^{v} = 7.292115 \times 10^{-5}$ rad / sec , earth rotation rate (+ccw about the north axis)

$g = 9.7803267715$ m / s^2 , equatorial value of gravity

a = 6388137 m , reference earth ellipsoid semiminor axis

e = 0.08181919 , earth eccentricity

$GM = 3.986005 \times 10^{14}$ m^3 / s^2 , gravitational mass constant

Coordinate Systems

Local Geodetic Vertical System

In order to make sense of the data output from the IMU sensors, a suitable coordinate system is selected. In the case of ground vehicle navigation, the position coordinates are normally in geodetic latitude, geodetic longitude and height (above mean sea level) and measured in the Local Geodetic Vertical (LGV) coordinate system. LGV coordinates are readily available when using GPS and is currently the standard used in the NTV. The LGV (sometimes called geographic) coordinate system is illustrated in Figure 5.18 [Cha97].

Other coordinates systems that are to be referred to in this text include Earth-Centered Inertial (ECI), Earth-Centered Earth-Fixed (ECEF), and Local Geocentric Vertical (LGCV).

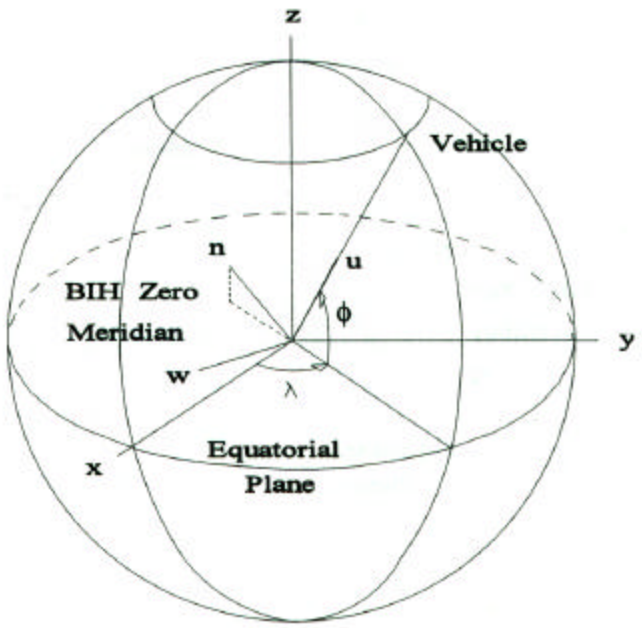


Figure 5.18: Geometry of LGV coordinate system.

where,

ϕ - geodetic latitude
λ - geodetic longitude
origin - center of mass of the Earth
n - axis in the direction of geodetic north
w - axis perpendicular to the meridian plane containing the vehicle, directed toward the west
u - axis directed outward along the local geodetic vertical passing through the vehicle (IMU ceneter of mass)

Local Geocentric Vertical (LGCV) coordinate system

The LGCV is similar to the LGV coordinate system except that the vertical axis is coincident with the local geocentric vertical. This coordinate system is denoted by the subscript or superscipt **c**. This is illustrated in Figure 5.19

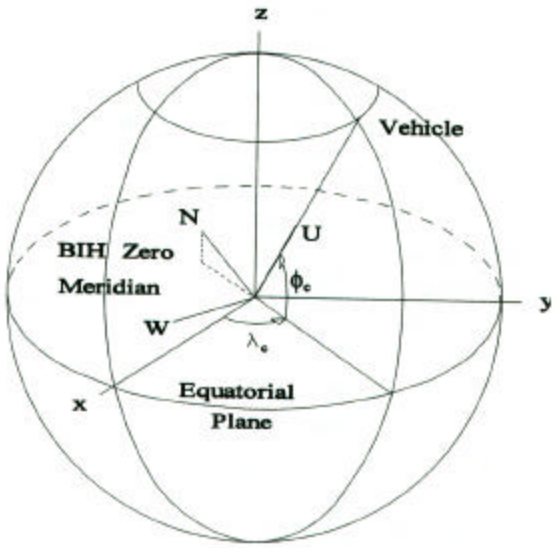


Figure 5.19: Geometry of LGCV coordinate system.

where,

ϕ_c - geocentric latitude
λ_c - geocentric longitude
origin - center of mass of the Earth
N - axis in the direction of geodetic north
W - axis perpendicular to the meridian plane containing the vehicle, directed toward the west
U - axis directed outward along the local geocentric vertical passing through the vehicle (IMU ceneter of mass)

Earth-Centered Inertial (ECI) coordinate system

The orientation of the coordinate axes is arbitrary. At any time, after navigation begins, the ECI coordinates remain in a fixed orientation in inertial space while the origin moves with the Earth. This coordinate system is denoted by the subscript or superscipt **i**.

Earth-Centered Earth-Fixed (ECEF) coordinate system

The ECEF coordinate system coincides with the conventional terrestrial reference system (CTRS). The origin is at the center of mass of the Earth. The coordinates remain fixed relative to the rotating Earth. The ECEF frame rotates relative to the ECI frame at

the rotation rate of the Earth, ω_{ie}. This coordinate system is denoted by the subscript or superscipt **e**.

x - axis in the mean astronomic equatorial plane orthogonal to the z axis and in the BIH zero meridian plane
y - axis in the mean astronomic equatorial plane, 90° east of the x-axis.
z - axis coincides with the conventional terrestrial pole (CTP, rotation axis of the WGS 84 ellipsoid)

Coordinate Rotation Matrices

In order to transform from one coordinate system to another, a rotation matrix is defined. The following rotation matrices are to be used in the following sections.

A. ECI to ECEF

$$R_i^e = \begin{pmatrix} \cos\omega_{ie}t & \sin\omega_{ie}t & 0 \\ -\sin\omega_{ie}t & \cos\omega_{ie}t & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.4}$$

B. ECI to LGV

$$R_i^v = \begin{pmatrix} -\cos\lambda^i \sin\phi & -\sin\lambda^i \sin\phi & \cos\phi \\ \sin\lambda^i & -\cos\lambda^i & 0 \\ \cos\lambda^i \cos\phi & \sin\lambda^i \cos\phi & \sin\phi \end{pmatrix} \tag{5.5}$$

where $\lambda^i = \lambda + \omega_{ie}t$

C. ECI to LGCV

$$R_i^c = \begin{pmatrix} -\cos\lambda^i \sin\phi_c & -\sin\lambda^i \sin\phi_c & \cos\phi_c \\ \sin\lambda^i & -\cos\lambda^i & 0 \\ \cos\lambda^i \cos\phi_c & \sin\lambda^i \cos\phi_c & \sin\phi_c \end{pmatrix} \tag{5.6}$$

where ϕ_c is the geocentric latitude

D. ECEF to LGV

$$R_e^v = \begin{pmatrix} -\cos\lambda\sin\phi & -\sin\lambda\sin\phi & \cos\phi \\ \sin\lambda & -\cos\lambda & 0 \\ \cos\lambda\cos\phi & \sin\lambda\cos\phi & \sin\phi \end{pmatrix} \quad (5.7)$$

Coordinate Axes Rotation

Normally, the convention for body and navigation coordinates are expressed as follows:

Vehicle (IMU) ← →	Body CS ← →	LGV CS
Forward	X	North
Right	Y	East
Down	Z	Down

However, the succeeding equations for calibration, coarse and fine alignment, and the navigation solution use a different convention and are based from Chatfield.

Vehicle (Nav) ← →	Body CS ← →	LGV CS
Forward	X	North
Left	Y	West
Up	Z	Up

Therefore, the IMU data is first manipulated once it is received on the PMP. The coordinate axes are rotated and the calibration coefficients are added in the following manner:

$$\text{Acc X} = (\text{Acc Out X}) + (\text{bias X}) \quad (5.8)$$

$$\text{Acc Y} = -(\text{Acc Out Y}) + (\text{bias Y}) \quad (5.9)$$

$$\text{Acc Z} = -(\text{Acc Out Z}) + (\text{bias Z}) \quad (5.10)$$

$$\text{Omega X} = (\text{Omega Out X}) + (\text{drift X}) \quad (5.11)$$

$$\text{Omega Y} = -(\text{Omega Out Y}) + (\text{drift Y}) \tag{5.12}$$

$$\text{Omega Z} = -(\text{Omega Out Z}) + (\text{bias Z}) \tag{5.13}$$

where,

Acc Out - output accelerations parsed from the IMU data message.

Omega Out - output angular velocities parsed from the IMU data message.

It should be remembered that the rotation of the body coordinate axes is only for the purpose of maintaining consistency with the convention used in the equations for alignment and navigation solution (Nav CS). However, once the position, velocity and attitude (PVA) have been calculated, the output PVA data will be converted back to the original body (IMU CS) coordinate system. As explained earlier, this puts the data back into the axes convention currently being used by the NTV.

Steps

1. Preliminary System Check

The first step is to ascertain if all the sensors are working properly. A quick check is to boot up the IMU and monitor the output. The key is to move the IMU exclusively in one direction (whether linearly along the axis or rotating it about an axis) and quickly see if the data is moving in the right direction. Take note of the x, y and z axes of the IMU. For the accelerometers, it is advisable to point each positive axis in the down direction with respect to the earth, making sure it is stationary and expect values close to the known gravity value (approximately 32 ft/sec^2).

2. Obtaining Known Reference

Next, a suitable known reference should be used. For reference position, one can survey in a point where the calibration will be performed. Here, calibration is performed

inside the CIMAR lab with known geodetic coordinates (Latitude = 29.646416 deg, Longitude = -82.349352 deg, Altitude = 12.34 m.). For reference orientation, either an Inertial Navigation System (INS) or an accurate compass can be used. In our application, the Honeywell MAPS is a stand-alone INS that gives highly accurate orientation. Setting up the MAPS inside the lab (see Figure 5.20), the IMU must then physically referenced with the MAPS, using its machined mounting surface as the planar guide. Initially, the MAPS must be aligned, then oriented such that roll, pitch and yaw values are zero (due North and level). By doing such, the extraction of the biases become much simpler.

Aside from the MAPS, other devices for accurately measuring orientation can be used. A digital compass can serve this purpose.

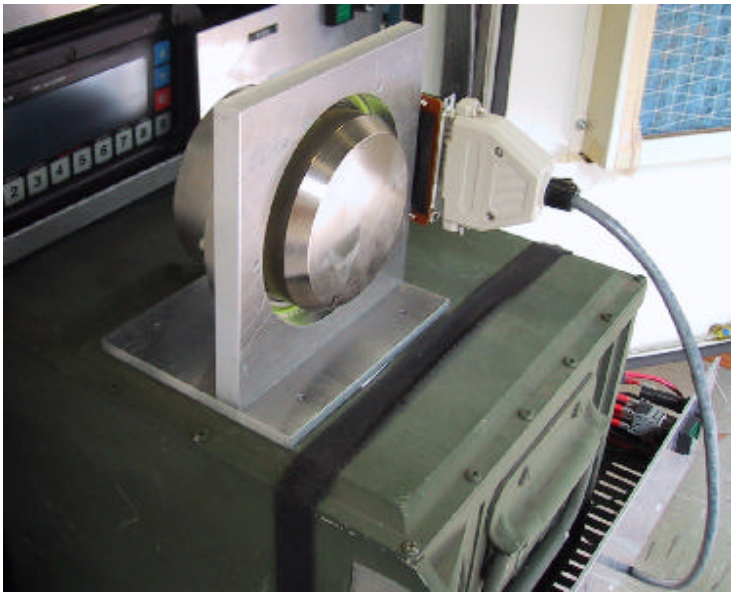


Figure 5.20: Using MAPS as Calibration Reference.

3. Performing Tumble Rotation Schedule

A Tumble Rotation Schedule consists of rotating the IMU in various fixed orthogonal positions (e.g. due North, due East, etc.) as well as mid positions (45 degrees

between 2 orthogonal axes) . A Tumble Platform allows one to precisely rotate the IMU axes. A complete tumble rotation schedule (31 point) will provide excellent calibration data. But in the absence of a tumble platform, the MAPS is again used as a reference. Due to limitations in calibration set-up, the IMU will only be rotated about its z-axis (yaw), orienting it in four positions (yaw = 0, 90, 180, 270 degrees). During each tumble rotation, static data will be collected for 60-120 seconds. A list of the tumble rotations and the data sets associated with them were made.

4. Calculating Calibration Coefficients

With the collected data, the calibration coefficients will be extracted. This is done by comparing the data with the true values for specific force (accelerations) and angular velocity at each orientation. The known values for specific force are based on the gravity vector while the known angular velocities are components of the earth rotation rate. For instance, with the z-axis of the IMU aligned with the z-axis of the MAPS, the true z value is +32.0 ft/sec^2 (or 9.8 m/sec^2). The coefficients are calculated by subtracting the data value from the true value.

$$\textit{Calib Coeff (Bias/Drift)} = \textit{Data Value} - \textit{True (Known) Value} \qquad (5.14)$$

When performed correctly and precisely, the calibration coefficients should be consistent for each sensor axis regardless of the orientation. But due to real-world imperfections, averaging the coefficients is sufficient enough for good results. In the case of the IMU, biases are normally in the range of 0.001 to 0.008 m/sec^2, while drifts are in the range of 0.00001 to 0.00008 rad/sec, which are consistent with specifications. Random bias values range from 0.005 to 0.01 m/sec^2 while gyro random drift values

range from 0.0001 to 0.0004 rad/sec. These are alarmingly high considering that they are 5 to 10 times larger than the fixed values. Therefore, manual calibration marginally improves the performance of the IMU but is still a necessary process. The detailed results are given below in Table 5.24.

Table 5.24: 4-Point Tumble Rotation Schedule.

Yaw	Drift (rad/sec)			Bias (m/sec^2)		
(Degrees)	X-axis	Y-axis	Z-axis	X-axis	Y-axis	Z-axis
0	-1.3754E-05	6.9595E-06	1.0389E-05	5.3854E-03	-5.6746E-03	-8.3522E-03
90	-1.4612E-05	4.9548E-06	7.7260E-06	-5.1076E-02	4.9301E-02	-7.3802E-03
180	-1.5618E-05	6.1590E-06	6.2344E-06	2.4916E-02	4.9491E-02	-7.3175E-03
270	-1.2671E-05	6.6012E-06	4.9628E-06	-1.0366E-02	-2.1958E-04	-7.3906E-03
Average	-1.4164E-05	6.1686E-06	7.3280E-06	-7.7851E-03	2.3224E-02	-7.6101E-03

5. Testing through Alignment

After calculating the six calibration coefficients, the easiest way to verify its accuracy is to do an analytical coarse alignment. Still using the MAPS as the reference, orient the IMU is oriented exactly the same way (roll, pitch and yaw equal zero). The coarse alignment code is then run. It should give roll and pitch error values in the range of ± 0.5 degrees, while the yaw error value of ± 3 degrees. Several coarse alignment runs were made until the system output appeared stable and accurate. One can also orient the IMU with different yaw values (e.g. x-axis pointed due north, yaw = +90 degrees) and check accuracy.

Alignment

Alignment of a strapdown IMU is the process of establishing the rotation matrix that relates the vehicle body coordinates and the navigation coordinates. In this case, the navigation coordinates refer to the Local Geodetic Vertical (LGV) coordinate system. Also, the vehicle body coordinates coincide with the roll, pitch and yaw axes.

The process of alignment is accomplished computationally by computing the elements of the rotation from the vehicle body axes to the navigation coordinates based on known reference vectors. After completion of the initial alignment and initialization of the state vector, the IMU is now ready for the navigation phase. (Cha97)

The essential output of the alignment phase is the initial rotation matrix from the body to navigation coordinates ($R_b^{v}{}_o$), which is one of the inputs (together with initial latitude, longitude and altitude) at the start of the navigation phase. During navigation, the rotation matrix is continuously updated based on the output of the gyros and accelerometers.

It is important to stress the role of alignment in the entire inertial navigation process. With proper instrument calibration, the accuracy of the alignment process ensures an excellent starting point for the navigation phase. Conversely, inaccurate alignment will lead to erroneous navigation.

There are two phases to alignment: the analytical coarse alignment and the fine alignment. The analytical coarse alignment of a strapdown inertial system consists of determining the rotation matrix that would convert the output of the gyros and accelerometers in body coordinates to navigation coordinates matching it to known reference vectors. The fine alignment phase uses the initial rotation matrix and static data fed it into a Kalman Filter to progressively eliminate misalignment errors. A summary of the process time for the entire alignment sequence is given in Table 5.25.

Table 5.25: Initial Alignment Process Breakdown.

Alignment Phase	Time (min)	Roll Accuracy (deg)	Pitch Accuracy (deg)	Yaw Accuracy (deg)
Coarse	2	± 0.5	± 0.5	± 1.0
Fine	6-8	± 0.2	± 0.2	± 0.5

For alignment to take place, a "Static" condition of the IMU must be satisfied. "Static" here refers to the IMU passing certain data conditions representing zero motion. Since the IMU is comprised of gyroscopes and accelerometers, the raw data output is passed through two checks, using threshold values set by the user.

Static Condition Determination

Since even slight IMU movement will cause misalignment errors, it is key to set data conditions that would best represent actual operation. To accomplish this, data was collected during various levels of IMU stability. Data was then analyzed as to the ranges they fall into, and their corresponding RMS (Root Mean Square) values.

In this case, the acceptable level of instability is the condition with the IMU on the Navigation Test Vehicle (NTV) with the engine, generator and all the other vehicle systems running and fall under the "vibrating mode". The condition where unwanted passenger or wind loading is experienced is termed "shaking mode". It is clear from Figure 5.22 that movement shifts the Omega RMS (ω_{RMS}) data curve upwards. However, Figure 5.23 shows that the Acceleration RMS (a_{RMS}) values have greater oscillation with IMU movement. A safe threshold value of 0.05 rad/sec is chosen for Omega RMS, falling in between "vibrating" and "shaking" conditions. For the Acceleration RMS, a threshold range of 9.76 to 9.84 m/sec^2 is set. For data to be considered "static", it must pass both checks.

Check 1 : $\omega_{RMS_{data}} < \omega_{RMS_{threshhold}} < 0.05 \ {}^{rad}/_{sec}$

Check 2 : $\omega_{RMS_{lower\_threshhold}} < a_{RMS_{data}} < \omega_{RMS_{upper\_threshhold}}$ ($9.76 \ {}^{m}/_{sec^2} < a_{RMS_{data}} < 9.84 \ {}^{m}/_{sec^2}$)

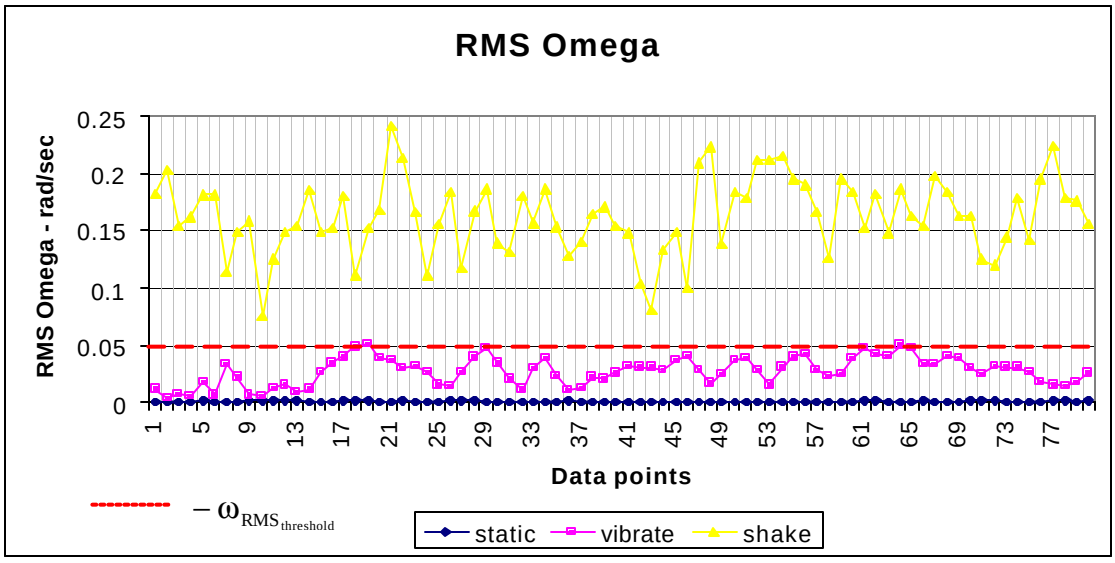


Figure 5.22: Omega RMS in Static, Vibrate and Shaking Mode.

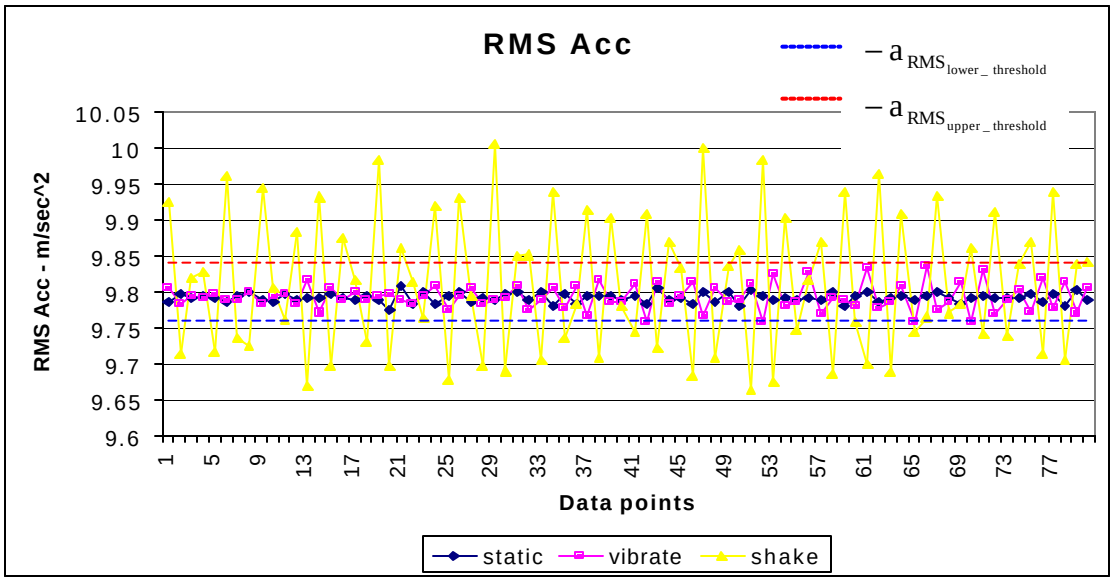


Figure 5.23: Acceleration RMS in Static, Vibrate and Shaking Mode.

Analytical Coarse Alignment

The first phase of alignment requires a simple approach and is computationally efficient. While the vehicle is stationary, static data is collected from the IMU. However, even during static conditions, brief or instantaneous movement may occur. A

movement limit of 3 consecutive seconds of non-static data resets the alignment process. Otherwise, static data is collected for 120 seconds. The collected data is then averaged.

The next step involves calculating the initial rotation matrix, $R^{v}_{b_0}$, from the following equation:

$$R^{v}_{b} = M^{-1} Q \tag{5.15}$$

where, R^{v}_{b} - rotation matrix from body to navigation coordinates
M - reference vector matrix

$$M = \begin{bmatrix} g_a^{v\,T} \\ \omega_{ie}^{v\,T} \\ (g_a^{v} \times \omega_{ie}^{v})^{T} \end{bmatrix} \tag{5.16}$$

where, g^{v}_{a} - actual apparent gravity vector in navigation coordinates

$$g^{v}_{a} = g - \omega^{v}_{ie} \times \omega^{v}_{ie} \times P^{v} \tag{5.17}$$

$$g = \begin{bmatrix} 0 \\ 0 \\ -9.801 \ m/s^2 \end{bmatrix} \tag{5.18}$$

ω^{v}_{ie} - Earth-rate vector

$$\omega^{v}_{ie} = \omega_{ie} \begin{bmatrix} +\cos f \\ 0 \\ +\sin f \end{bmatrix} \tag{5.19}$$

where $\omega_{ie} = 7.292115 \times 10^{-5}$ rad/s
- Earth rotation rate (+ ccw about north)
ϕ - latitude

Q - IMU data matrix

$$Q = \begin{bmatrix} g_b^T \\ \omega_b^T \\ (g_b \times \omega_b)^T \end{bmatrix} = \begin{bmatrix} -S_b^T \\ \omega_b^T \\ (-S_b \times \omega_b)^T \end{bmatrix} \tag{5.20}$$

where, S_b - accelerometer cluster output in body coordinates

$$S_b = \begin{bmatrix} S_{x_b} \\ S_{y_b} \\ S_{z_b} \end{bmatrix} \tag{5.21}$$

ω_b - gyroscope output in body coordinates

$$\omega_b = \begin{bmatrix} \omega_{x_b} \\ \omega_{y_b} \\ \omega_{z_b} \end{bmatrix} \tag{5.22}$$

Substituting,

$$Q = \begin{bmatrix} -S_{x_b} & -S_{y_b} & -S_{z_b} \\ \omega_{x_b} & \omega_{y_b} & \omega_{z_b} \\ S_{z_b}\omega_{y_b} - S_{y_b}\omega_{z_b} & S_{x_b}\omega_{z_b} - S_{z_b}\omega_{x_b} & S_{y_b}\omega_{x_b} - S_{x_b}\omega_{y_b} \end{bmatrix} \tag{5.23}$$

The accelerometer and gyroscope output are already adjusted using the calibration coefficients previously determined. Once the rotation matrix has been calculated, the orientation angles for roll, pitch, and yaw can be extracted. The rotation matrix is then passed on to the second phase, the fine alignment.

Since,

$$R_b^v = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi - \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi - \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{5.24}$$

The orientation angles are expressed as follows:

$$\text{roll, } \phi = \tan^{-1}\left(R^{v}_{b\ 3,2} / R^{v}_{b\ 3,3} \right) \tag{5.25}$$

$$\text{pitch, } \theta = \sin^{-1}\left(- R^{v}_{b\ 3,1} \right) \tag{5.26}$$

$$\text{yaw, } \psi = \tan^{-1}\left(R^{v}_{b\ 2,1} / R^{v}_{b\ 1,1} \right) \tag{5.27}$$

Fine Alignment

Because of the imperfect accelerometer and gyro outputs used in the analytical alignment computations, an additional dynamic second-stage alignment computation procedure is generally required. The output of the three accelerometers and three gyros can be used to provide a computational self-alignment procedure to reduce the difference between the actual and analytically computed rotation [Cha97].

Error Equation Derivation

The fine alignment procedure is based on Britting and Palsson [Bri70] with the variables modified to maintain consistency with the navigation convention used by Chatfield [Cha97]. It is assumed here that an initial estimate of the rotation matrix is available from the analytical coarse alignment that corresponds to a small angle misalignment of the computed and actual reference frame. Fine or corrective alignment consists of detecting the error angles between these two frames via the processed accelerometer and gyro signals and generating a signal to the transformation computer in order to reduce these angles as close to zero as possible. The rotation matrix is updated using the relation

$$\dot{R}^{v}_{b} = R^{v}_{b} \Omega^{b}_{vb} \tag{5.28}$$

where Ω_{vb}^{b} is the skew-symmetric matrix of the angular velocity ω_{vb}^{b} that is fed to the attitude computer. This angular velocity signal would ideally be

$$\omega_{vb}^{b} = \underline{\omega}_{v}^{b} + \omega_{d}^{b} \tag{5.29}$$

where $\underline{\omega}_{v}^{b}$ is the computed correction signal (or command rate)

ω_{d}^{b} is the disturbance represented by vibrations in the body frame

$$\omega_{d}^{b} = \omega_{d}^{b} - E^{b}\omega_{ie}^{b} + u\omega^{b} \tag{5.30}$$

where $u\omega^{b}$ contains the gyro drift

E^{b} is defined as the skew symmetric matrix of the misalignment angles between the actual and computed body frames.

$$E^{b} = \begin{bmatrix} 0 & -\varepsilon_{z} & \varepsilon_{y} \\ \varepsilon_{z} & 0 & -\varepsilon_{x} \\ -\varepsilon_{y} & \varepsilon_{x} & 0 \end{bmatrix} \tag{5.31}$$

where ε_{x}, ε_{y}, and ε_{z} are the misalignment angles in the body x, y, and z axes

Substituting the skew symmetric form of Eq. (5.30) into Eq. (5.28) yields

$$\dot{R}_{b}^{v} = R_{b}^{v}\underline{\Omega}_{v}^{b} + R_{b}^{v}\Omega_{d}^{b} - R_{b}^{v}\delta\Omega_{ie}^{b} + R_{b}^{v}u\Omega^{b} \tag{5.32}$$

where $\delta\Omega_{ie}^{b}$ is the skew symmetric form of $E^{b}\omega_{ie}^{b}$

After manipulation, Eq. (5.30) simplifies to

$$\dot{R}_{v}^{v}R_{b}^{v} = R_{b}^{v}\underline{\Omega}_{v}^{b} - R_{b}^{v}\delta\Omega_{ie}^{b} + R_{b}^{v}u\Omega^{b} \tag{5.33}$$

Using the fact that $R_{\tilde{v}}^{v} = I - E^{v}$ and $\dot{E}^{v} = R_{b}^{v}E^{b}R_{b}^{v}$, Eq. (5.33) reduces to

$$\dot{E}^{b} = -\underline{\Omega}_{v}^{b} - u\Omega^{b} + \delta\Omega_{ie}^{b} \tag{5.34}$$

$$\dot{E}^{v} = -\underline{\Omega}_{v}^{v} - u\Omega^{v} + \delta\Omega_{ie}^{v} \tag{5.35}$$

The vector form of Eq. 5.35 is

$$\dot{\varepsilon}^v + \underline{\omega}_v^{vv} = -u\omega^v - \Omega_{ie}^v \varepsilon^v \tag{5.36}$$

In order to drive E^v to zero, $\underline{\omega}_v^v$ is chosen to be a linear function of E^v.

$$\dot{\varepsilon}^v + K\tilde{\varepsilon}^v = -u\omega^v - \Omega_{ie}^v \varepsilon^v \tag{5.37}$$

where $\tilde{\varepsilon}^v = \varepsilon^v + \delta\varepsilon$, is the misalignment measurement error

In order to measure $\tilde{\varepsilon}^v$, its three components can be derived from the horizontal components of g and the computed west component of ω_{ie}^v.

$$\text{since} \quad \underline{E}^v = \begin{bmatrix} 0 & -\underline{\varepsilon}_u & \underline{\varepsilon}_w \\ \underline{\varepsilon}_u & 0 & -\underline{\varepsilon}_n \\ -\underline{\varepsilon}_w & \underline{\varepsilon}_n & 0 \end{bmatrix} \tag{5.38}$$

$$\text{and} \quad \underline{S}^v = R_b^v \underline{S}_c^b = (I - \underline{E}^v) R_{b0}^v \underline{S}_c^b = (I - \underline{E}^v) R_{b0}^v [\underline{S}^{\tilde{v}} + \underline{S}_d^v + u\underline{S}^v] \tag{5.39}$$

where $\underline{S}^v = [0;0;g]$

$\underline{S}_d^v$ is the disturbance accelerations

uS^v is the accelerometer uncertainties

R_{b0}^v is the initial rotation matrix

Combining S_d^v and uS^v into $\delta\underline{S}^v$, and substituting Eq.(5.38) into Eq. (5.39)

$$(I - \underline{E}^v)\underline{S}^v = \underline{S}^{\tilde{v}} \tag{5.40}$$

$$\underline{S}^v - \underline{E}^v \underline{S}^v = \underline{S}^{\tilde{v}} \tag{5.41}$$

$$\underline{E}^v \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} = \underline{S}^{\tilde{v}} - \underline{S}^v = \Delta\underline{S}^{\tilde{v}} \tag{5.42}$$

$$\Delta\underline{S}_n = -\underline{g}_a \underline{\varepsilon}_w + \delta\underline{S}_n \approx -\underline{g}_a \underline{\varepsilon}_w \tag{5.43}$$

$$\Delta\underline{S}_w = \underline{g}_a \underline{\varepsilon}_n + \delta\underline{S}_w \approx \underline{g}_a \underline{\varepsilon}_n \tag{5.44}$$

Similarly, using the equation for computed west rate of the Earth rate,

$$\Delta\underline{\omega}_w = \omega_{ie}\cos\phi(\underline{\varepsilon}_u + \underline{\varepsilon}_n \tan\phi) + \delta\underline{\omega}_w \approx \omega_{ie}\cos\phi(\underline{\varepsilon}_u + \underline{\varepsilon}_n \tan\phi) \tag{5.45}$$

Given Eqs. (5.43), (5.44) and (5.45),

$$\underline{\varepsilon}_n \approx \frac{\Delta\underline{S}_w}{\underline{g}_a} \tag{5.46}$$

$$\underline{\varepsilon}_w \approx -\frac{\Delta\underline{S}_n}{\underline{g}_a} \tag{5.47}$$

$$\underline{\varepsilon}_u \approx \frac{\Delta\underline{S}_w}{\underline{g}_a}\tan\underline{\phi} - \frac{\Delta\underline{\omega}_w}{\underline{\omega}_{ie}}\sec\underline{\phi} \tag{5.48}$$

The error equation can be put into the form

$$\dot{\underline{\varepsilon}} = -\Omega^v_{ie}\underline{\varepsilon}^v - K\tilde{\varepsilon}^v = -\Omega^v_{ie}\underline{\varepsilon}^v - K\underline{\varepsilon}^v - K(\tilde{\varepsilon}^v - \underline{\varepsilon}^v) = -(\Omega^v_{ie} + K)\underline{\varepsilon}^v - K(\tilde{\varepsilon}^v - \underline{\varepsilon}^v) \tag{5.49}$$

which is in the general form, $\dot{x} = Fx + u$

Data Pre-Filtering

In order to eliminate the effects of data noise, pre-filtering becomes a necessary step before calculating the misalignment errors. Using a stripped-down version of a Kalman filter with a steady-state Kalman gain matrix, the output data is smoothened out eliminating unwanted spikes. The Kalman gain matrix is determined as a function of the signal to noise ratio and is specific to the sensor outputs. This kind of Kalman filter is termed a scalar-gain interpretation.

$$\tilde{\underline{S}}_b^- = \Phi_{sgi}\tilde{\underline{S}}_b \tag{5.50}$$

$$\tilde{\underline{S}}_b = \tilde{\underline{S}}_b^- + K_{pf_s}(\underline{S}_b - \tilde{\underline{S}}_b^-) \tag{5.51}$$

where $\tilde{\underline{S}}_b^-$ is the previous state estimate of the data

$\underline{S}_b$ is the current state estimate of the data

$\Phi_{sgi} = I$, is the state transition matrix since the model = 0

K_{pf_s} is the pre-filter Kalman gain matrix for the accelerometer output

$$K_{pf_s} = \begin{bmatrix} K_s & 0 & 0 \\ 0 & K_s & 0 \\ 0 & 0 & K_s \end{bmatrix} \tag{5.52}$$

The system was tuned and K_s was determined to equal 0.05.

Thus, the pre-filtered output of the accelerometer data is given by $\tilde{\underline{S}}_b$. The next iteration uses its last $\tilde{\underline{S}}_b$ and assigns it to the past state estimate, $\tilde{\underline{S}}_b^-$, and the whole process is repeated. Similarly, the same equations apply for pre-filtering the output of the gyroscopes. It should be noted that the Kalman gain matrix may not necessarily be the same for both sensors. In the case of the IMU, the Kalman gain matrices used are the same. Once the data has been pre-filtered, it can now be used in a Kalman filter that determines the error angles. Figure 5.23 shows the raw data and pre-filtered data outputs using the accelerations measured in the Z axis.

$$\tilde{\underline{\omega}}_b^- = \Phi_{sgi}\tilde{\underline{\omega}}_b \tag{5.53}$$

$$\tilde{\underline{\omega}}_b = \tilde{\underline{\omega}}_b^- + K_{pf_\omega}(\underline{\omega}_b - \tilde{\underline{\omega}}_b^-) \tag{5.54}$$

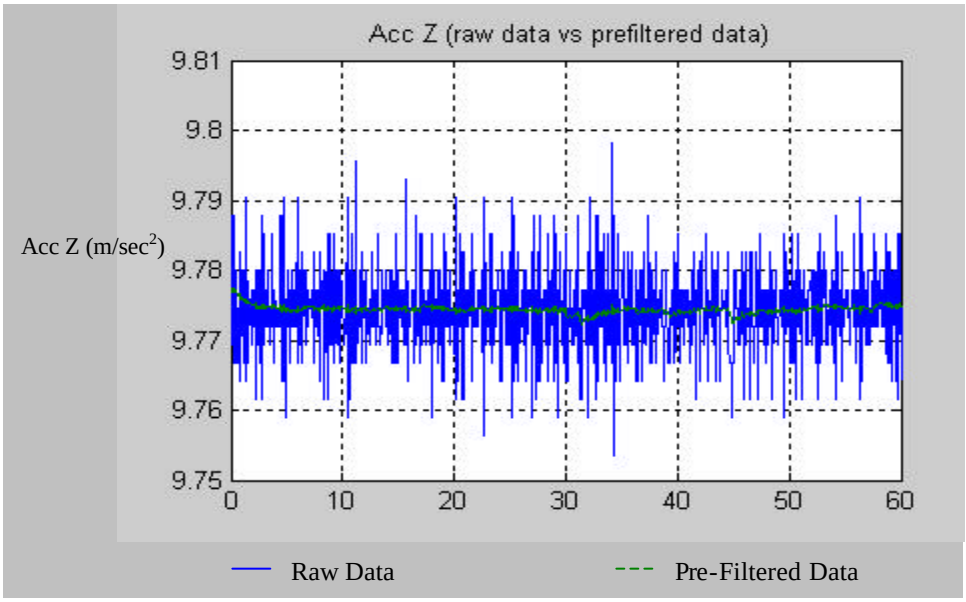


Figure 5.23: IMU Raw Data vs. Pre-filtered Data.

Error Angle Matrix Calculation

Eq. (5.36) is the error angle matrix, $\underline{E}^v$. A Kalman filter uses the error differential equation to solve for the optimal estimates. By selecting an arbitrary value for the error covariance, R, then using the closed-form solution of the Ricatti equation, the Kalman gain matrix, K, is calculated. And by discretizing $(\Omega_{ie}^v - K)$ and K, we get Φ and K_d. These calculations are done off-line using Matlab and later plugged into the PMP code in order to make the fine alignment process more computationally efficient. The constant values for Φ and K_d can now be used to determine the error angle matrix, $\underline{E}^v$, by letting the error angles converge to certain values.

$$\underline{S}^v = (I - \underline{E}^v)R_{b_0}^v \tilde{S}^b \qquad (5.55)$$

where $R_{b_0}^v$ is the initial rotation matrix output from coarse alignment

$$\underline{Z} = \begin{bmatrix} \underline{S}^v / g \\ \underline{\omega}^v / g \\ (\underline{S}^v / g) \times (\underline{\omega}^v / g) \end{bmatrix} \tag{5.56}$$

$$\underline{e}^{v^-} = \Phi \underline{e}^v \tag{5.57}$$

where $\underline{e}^v = \begin{bmatrix} \varepsilon_n \\ \varepsilon_w \\ \varepsilon_u \end{bmatrix}$ is the error angle vector

$$\tilde{\underline{Z}} = \underline{e}^{v^-} \tag{5.58}$$

$$K_\Delta = K_d(\underline{Z} - \tilde{\underline{Z}}) \tag{5.59}$$

$$\underline{e}^v = \underline{e}^{v^-} + K_\Delta \tag{5.60}$$

Using Eq. (5.55), we build the error angle matrix, $\underline{E}^v$. The error angles converge to certain values. The speed and accuracy of convergence is dependent on the Q selected. Finally, the resulting rotation matrix after fine alignment, $R^{v\prime}_{b_O}$ is equal to:

$$R^{v\prime}_{b_0} = (I - \underline{E}^v) R^v_{b_0} \tag{5.61}$$

This new initial rotation matrix is then passed on as the initial orientation condition entering the navigation phase. Using Eqs. (5.25), (5.26), and (5.27), we obtain converged values for roll, pitch, and yaw. Based on specifications and preliminary tests, typical convergence values are to within ± 1.0 deg for the yaw and ± 0.5 deg for the roll and pitch. Given ideal static conditions, the fine alignment stage runs from 6-8 minutes.

Navigation Solution

Once the initial rotation matrix has been determined through the two stage alignment process and the calculated calibration coefficients, inertial navigation can begin. The navigation solution uses initial position, velocity and attitude (PVA) inputs to initialize the states, and propagates these states using the measurements from the IMU sensors.

Local Geodetic Vertical (LGV) coordinates are convenient for land-based navigation because the velocity in the state vector is equivalent to the ground speed vector. The equations of motion are derived from a pair of vector differential equations of position and velocity. The equations are then put into state-space form.

Navigation is divided into two components: updates for (1) position and velocity (PV), and (2) attitude (A). The PV states are directly affected by the rotation matrix (or direction cosine matrix), R_b^v. However, using the gyro outputs, the rotation matrix is propagated independently from position and velocity. The block diagram of the LGV frame mechanization is shown in Figure 5.24 [Tit97].

In practice, the navigation computer, attitude computer and gravity computer are all part of the Navigation Processor (NP). In the IMU/GPS, the NP is one of the major blocks of the POS Main Processor (PMP). Similarly, the accelerometer and gyro measurements are fed into the NP together and are processed by separate parts of the navigation code.

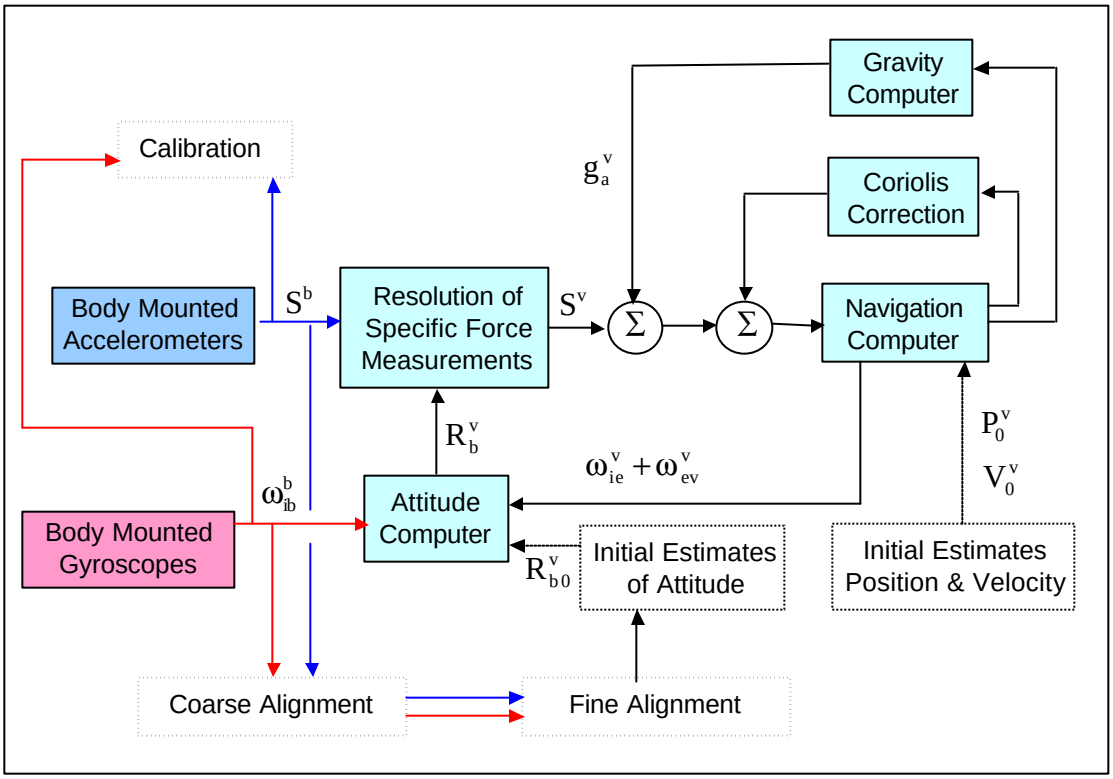


Figure 5.24: Local Geodetic Vertical (LGV) Frame Mechanization.

Position and Velocity

First, all the succeeding equations are taken from Chatfield, et al. [Cha97]. The concept lies on resolving the measured specific forces by the accelerometers in the navigation frame (LGV), integrating it once to give velocity and twice to give position. In order to set-up the equations in LGV, the state vectors are defined in ECEF coordinates. The position and velocity vectors are denoted by P and V respectively.

$$P^e = R_i^e P^i \tag{5.62}$$

solving for P^i and differentiating with respect to time

$$\dot{P}^i = \Omega_{ie}^e R_e^i P^e + R_e^i \dot{P}^e \tag{5.63}$$

note that $\dot{R}_i^e = \Omega_{ie}^e R_e^i$ and Ω_{ie}^e is the skew-symmetric form of ω_{ie}^e

a second differentiation with respect to time yields

$$\ddot{P}^i = R_e^i \left(\ddot{P}^e + 2\Omega_{ie}^e \dot{P}^e + \Omega_{ie}^e \Omega_{ie}^e P^e \right) \tag{5.64}$$

next, substituting (5.59) into $\ddot{P}^i = g^i + S^i$

$$\ddot{P}^e + 2\Omega_{ie}^e \dot{P}^e + \Omega_{ie}^e \Omega_{ie}^e P^e = g^e + S^e, \text{ where } S^e \text{ is the accelerometer output} \tag{5.65}$$

or in terms of vector cross products

$$\ddot{P}^e + 2\omega_{ie}^e \times \dot{P}^e + \omega_{ie}^e \times \omega_{ie}^e \times P^e = g^e + S^e \tag{5.66}$$

now transforming it into LGV coordinates

$$P^v = R_e^v P^e \tag{5.67}$$

$$V^v = R_e^v \dot{P}^e \tag{5.68}$$

differentiating with respect to time and substituting into (5.64)

$$R_i^v \ddot{P}^i = \dot{V}^v + (\Omega_{iv}^v + \Omega_{ie}^v) V^v + \Omega_{ie}^v \Omega_{ie}^v P^v \tag{5.69}$$

$$\dot{V}^v + (\Omega_{iv}^v + \Omega_{ie}^v) V^v + \Omega_{ie}^v \Omega_{ie}^v P^v = R_c^v g_{SHC}^c + R_e^v g_{mod}^e + R_b^v S^b \tag{5.70}$$

substituting (5.67) into derivative of P^v with respect to time

$$\dot{P}^v = V^v - \Omega_{ev}^v P^v \tag{5.71}$$

(5.70) and (5.71) form the state-space form for LGV coordinates

The position vector, P^e, is given by

$$P^e = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \approx - \frac{GM}{P^2} \begin{pmatrix} (R_N + H)\cos\lambda\cos\phi \\ (R_N + H)\sin\lambda\cos\phi \\ (R_N(1-e^2) + H)\sin\phi \end{pmatrix} \tag{5.72}$$

Substituting Eq. (5.6) & Eq. (5.72) into (5.67) yields the position vector in LGV coordinates.

$$P^v = \begin{pmatrix} P_n \\ P_w \\ P_u \end{pmatrix} = \begin{pmatrix} -R_N e^2 \sin\phi\cos\phi \\ 0 \\ R_N(1-e^2\sin^2\phi)+H \end{pmatrix} \tag{5.73}$$

Equation (5.68) provides no information on longitude because the position vector is completely determined by two components lying on the longitudinal plane. Consequently, a separate integration must be performed to obtain λ. The geometric representation of P^v is shown in Figure 5.25.

Expressions for prime vertical radius or curvatures (or great normal), R_N, and the meridian radius, R_M, are given below.

$$R_N = \frac{a}{(1-e^2\sin^2\phi)^{\frac{3}{2}}} \tag{5.74}$$

$$R_M = \frac{a(1-e^2)}{(1-e^2\sin^2\phi)^{\frac{1}{2}}} \tag{5.75}$$

$$R_N = R_M \frac{(1-e^2\sin^2\phi)}{1-e^2} \tag{5.76}$$

Expanding Eq. (5.68) yields

$$V^v = \begin{pmatrix} V_n \\ V_w \\ V_u \end{pmatrix} = \begin{pmatrix} (R_M+H)\dot{\phi} \\ -(R_N+H)\dot{\lambda}\cos\phi \\ \dot{H} \end{pmatrix} \tag{5.77}$$

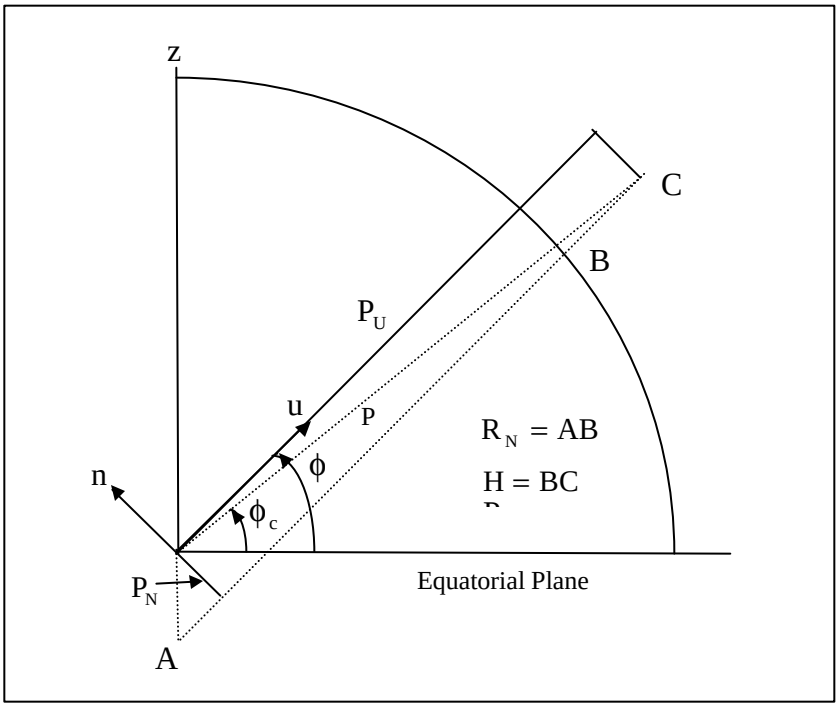


Figure 5.25: Components of position in LGV coordinates.

Using Eq. (5.77), we derive the expressions for geodetic latitude, longitude and height (altitude).

$$H_{k+1} = H_k + \int_{t_k}^{t_{k+1}} V_u \, d\tau \,, \text{ geodetic height} \tag{5.78}$$

$$\phi_{k+1} = \phi_k + \int_{t_k}^{t_{k+1}} \frac{V_n}{R_M + H} \, d\tau \,, \text{ geodetic latitude} \tag{5.79}$$

$$\lambda_{k+1} = \lambda_k - \int_{t_k}^{t_{k+1}} \frac{V_w \sec\phi}{R_N + H} \, d\tau \,, \text{ geodetic longitude} \tag{5.80}$$

Using Figure 5.25, an expression is derived for the rotation matrix from LGCV to LGV, R_c^v.

$$R_c^v = \begin{pmatrix} \cos(\phi - \phi_c) & 0 & -\sin(\phi - \phi_c) \\ 0 & 1 & 0 \\ \sin(\phi - \phi_c) & 0 & \cos(\phi - \phi_c) \end{pmatrix} \tag{5.81}$$

$$R_c^v = \frac{1}{P}\begin{pmatrix} P_u & 0 & P_n \\ 0 & 1 & 0 \\ -P_n & 0 & P_u \end{pmatrix} \tag{5.82}$$

It can also be seen from Figure 5.25 that the Earth rotation rate in LGV coordinates is given by Eq. (5.19).

The angular rate of the position vector is defined as the tangential velocity divided by the magnitude of the position vector. Hence, using Eq. (5.77)

$$\omega_{ev}^v = \begin{pmatrix} \dot{\lambda}\cos\phi \\ \dot{\phi} \\ \dot{\lambda}\sin\phi \end{pmatrix} = \begin{pmatrix} -\dfrac{V_w}{R_N + H} \\ \dfrac{V_n}{R_m + H} \\ -\dfrac{V_w}{R_N + H}\tan\phi \end{pmatrix} \tag{5.83}$$

Adding Eqs. (5.19) and (5.78) we get

$$\omega_{iv}^v = \omega_{ie}^v + \omega_{ev}^v \tag{5.84}$$

Combining Eqs. (5.65) and (5.66) into state-space form yields

$$\begin{pmatrix} \dot{V}^v \\ \dot{P}^v \end{pmatrix} = \begin{pmatrix} -(\Omega_{iv}^v + \Omega_{ie}^v) & -\Omega_{ie}^v\Omega_{ie}^v \\ I & -(\Omega_{iv}^v - \Omega_{ie}^v) \end{pmatrix}\begin{pmatrix} V^v \\ P^v \end{pmatrix} + \begin{pmatrix} R_c^v & R_e^v & R_b^v \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} g_{SHC}^c \\ g_{mod}^e \\ S^b \end{pmatrix} \tag{5.85}$$

The inertial navigation system block diagram is shown in Figure 5.26 [Cha97]. The differential equation as represented in state-space is given by:

$$\dot{X} = AX + BU \tag{5.86}$$

$$Y = CX \tag{5.87}$$

where,

$$\dot{X} = \begin{pmatrix} \dot{V}^v \\ \dot{P}^v \end{pmatrix}$$

$$A = \begin{pmatrix} -(\Omega_{iv}^v + \Omega_{ie}^v) & -\Omega_{ie}^v \Omega_{ie}^v \\ I & -(\Omega_{iv}^v - \Omega_{ie}^v) \end{pmatrix}$$

$$X = \begin{pmatrix} V^v \\ P^v \end{pmatrix}$$

$$B = \begin{pmatrix} R_c^v & R_e^v & R_b^v \\ 0 & 0 & 0 \end{pmatrix}$$

$$U = \begin{pmatrix} g_{SHC}^c \\ g_{mod}^e \\ S^b \end{pmatrix}$$

$$C = I$$

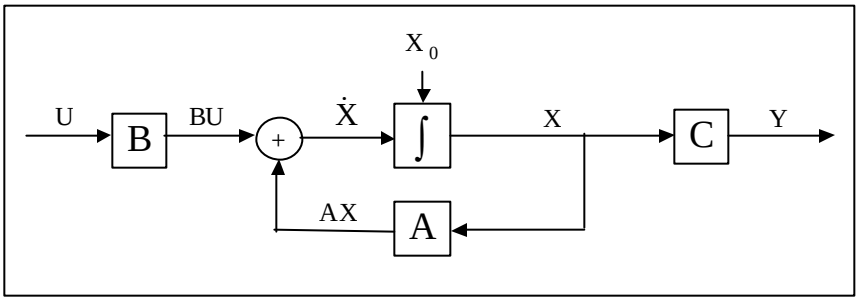


Figure 5.26: Inertial Navigation System Block Diagram.

The above differential equation can be solved using a fourth-order Runge-Kutta integration method. This process is outlined in Appendix A.

Orientation

The new orientation angles are a function of the current rotation matrix, R_b^v. The R_b^v matrix defines the rotation that is used transform the specific force vector, S^b, into navigation (LGV) coordinates, S^v. This matrix propagates in accordance with the following equation.

$$\dot{R}_b^v = R_b^v \Omega_{vb}^b \tag{5.88}$$

where Ω_{vb}^b is the skew-symmetric form of ω_{vb}^b, the body rate in LGV coordinates. This is derived by subtracting the estimates of the components of navigation frame rate, ω_{iv}^v, from the measured body rates, ω_{ib}^b. In Eq. (5.74), ω_{iv}^v is obtained by summing the Earth's rotation rate with respect to ECI frame and the turn rate of the navigation frame with respect to the Earth. Substituting this yields

$$\omega_{vb}^v = \omega_{ib}^b - R_v^b (\omega_{ie}^v + \omega_{ev}^v) \tag{5.89}$$

The skew-symmetric form of ω_{vb}^v is given by

$$\Omega_{vb}^b = \begin{pmatrix} 0 & -\omega_{z_b} & \omega_{y_b} \\ \omega_{z_b} & 0 & -\omega_{x_b} \\ -\omega_{y_b} & \omega_{x_b} & 0 \end{pmatrix} \tag{5.90}$$

where,

$$\omega_{vb}^v = \begin{pmatrix} \omega_{x_b} \\ \omega_{y_b} \\ \omega_{z_b} \end{pmatrix} \tag{5.91}$$

In order to propagate the rotation matrix, R_b^v, quaternions will used. The quaternion attitude is a four-parameter representation based on the idea that a transformation from one

coordinate frame to another may be effected by a single rotation about a vector [Tit97]. Quaternions are computationally efficient since they do not involve any trigonometric functions and they follow a product rule for successive rotations. However, the major disadvantage of quaternions is the lack of intuitive physical meaning.

The individual quaternion elements may be derived using the orientation angles extracted from the rotation matrix. Roll (ϕ), pitch (θ) and yaw (ψ) values are calculated using Eqs. (5.25), (5.26), and (5.27).

$$q_1 = \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \tag{5.92}$$

$$q_2 = \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \tag{5.93}$$

$$q_3 = \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \tag{5.94}$$

$$q_4 = \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} \tag{5.95}$$

Once the quaternion vector has been calculated, the differential equation is set-up as follows:

$$\begin{aligned}
\dot{q}_1 &= -0.5(b\omega_{x_b} + c\omega_{y_b} + d\omega_{z_b}) \\
\dot{q}_2 &= 0.5(a\omega_{x_b} - d\omega_{y_b} + c\omega_{z_b}) \\
\dot{q}_3 &= 0.5(d\omega_{x_b} + a\omega_{y_b} - b\omega_{z_b}) \\
\dot{q}_4 &= -0.5(c\omega_{x_b} - b\omega_{y_b} - a\omega_{z_b})
\end{aligned} \tag{5.96}$$

Similarly, the above differential equation will be solved using a fourth-order Runge-Kutta integration method. This is also described in Appendix A.

Once the new quaternion state has been determined, the elements are used to calculate the new rotation matrix, $R_b^{v'}$. The matrix representation is:

$$R_b^v = \begin{pmatrix} (q_1^2 + q_2^2 - q_3^2 - q_4^2) & 2(q_2q_3 - q_1q_4) & 2(q_2q_4 + q_1q_3) \\ 2(q_2q_3 + q_1q_4) & (q_1^2 - q_2^2 + q_3^2 - q_4^2) & 2(q_3q_4 + q_1q_2) \\ 2(q_2q_4 - q_1q_3) & 2(q_3q_4 + q_1q_2) & (q_1^2 - q_2^2 - q_3^2 + q_4^2) \end{pmatrix} \quad (5.97)$$

It should be noted that all the quaternion values used in (5.92) are taken from the results of the quaternion integration and thus define the new quaternion vector. The new R_b^v will then be used calculate the new position and velocity on the next data time interval.

An important check is maintaining orthogonality of the rotation matrix at all times. Non-orthogonality leads to numerical errors and consequent degrading of orientation, position and velocity calculation. To ensure orthogonality, a simple process of extracting the orientation angles using Eqs. (5.25), (5.26), and (5.27) is followed by putting those same angles back into Eq. (5.24) to get the orthogonalized rotation matrix.

DGPS Aiding and Extended Kalman Filter

The role of differential GPS (DGPS) position aiding in improving INS/GPS integrated system using low to medium performance IMUs cannot be understated. In the case of the IMU/GPS system, the position updates, when available, bounds the position, velocity and orientation errors allowing successful long-term operation.

One of the goals in INS testing is to determine the influence of sensor and initialization errors on system position and velocity accuracy. This is extremely difficult if a wide variety of different types of measurements are used. A Kalman filter is a natural solution to this problem

because it provides a statistical weighting and error propagation algorithm. Following an external "fix", the filter updates position and velocity on the basis of statistically weighting the covariance of the external measurement against the covariance of the estimated state of the system. Between updates the filter propagates estimates of the navigation solution and estimates of modeled sensors. Thus the KF is an intuitive approach that supplies an optimal estimate of the navigation system errors and provides estimates of all modeled navigation sensor error sources that have significant correlation times [Bie99].

Differential GPS (DGPS) with its high accuracy now becomes the source of position updates. The DGPS position updates are made to stored "waypoints" used by the KF to compute and apply corrections to the navigation equations. These position updates reduce attitude error as well as removing position error, thus slowing error growth between measurements. Velocity is used by the KF to update the range rate predicted by the navigation equations and thus to prevent buildup of position errors [Bie99].

The Kalman Filter used for the IMU/GPS is the same linear discrete KF used by the MAPS/GPS [Wit96]. It would have been advisable to use a KF with error equations that are tailored to the raw outputs of the accelerometers and gyros. However, the existing KF performs the exact same function effectively by eliminating errors through constant updates of the position, velocity and orientation in LGV coordinates.

We define an augmented state vector for the kth integration step that includes errors in position, position rate and tilt or platform attitude. The general equation for the state vector error is given by

$$\Delta\tilde{\underline{X}}_k = \Delta\tilde{\underline{X}}_k^- + K_k(\underline{Z}_k - \tilde{\underline{Z}}_k^-) \tag{5.98}$$

where

$$\Delta\tilde{\underline{X}}_k = \begin{bmatrix} \delta\phi \\ \delta\lambda \\ \delta H \\ \delta\dot{\phi} \\ \delta\dot{\lambda} \\ \delta\dot{H} \\ \phi_n \\ \phi_e \\ \phi_d \end{bmatrix}$$, estimated state vector error after the KF update

$\Delta\tilde{\underline{X}}_k^-$, estimated state vector error just before the KF update

$\tilde{\underline{Z}}_k^- = H_k \Delta X_k^-$, predicted measurement just before the KF update

$$\underline{Z}_p = \begin{bmatrix} \phi_{GPS} - \phi_{IMU} \\ \lambda_{GPS} - \lambda_{IMU} \\ H_{GPS} - H_{IMU} \end{bmatrix} = \begin{bmatrix} \phi_{GPS} - (\phi + \delta\tilde{\phi}) \\ \lambda_{GPS} - (\lambda + \delta\tilde{\lambda}) \\ H_{GPS} - (H + \delta\tilde{H}) \end{bmatrix}$$

$$H_p = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\underline{Z}_{\Delta p} = \begin{bmatrix} \Delta\phi_{GPS} - \Delta\phi_{IMU} \\ \Delta\lambda_{GPS} - \Delta\lambda_{IMU} \\ \Delta H_{GPS} - \Delta H_{IMU} \end{bmatrix} = \begin{bmatrix} \Delta\phi_{GPS} - (\Delta\phi + \delta\tilde{\dot{\phi}}\Delta T) \\ \Delta\lambda_{GPS} - (\Delta\lambda + \delta\tilde{\dot{\lambda}}\Delta T) \\ \Delta H_{GPS} - (\Delta H + \delta\tilde{\dot{H}}\Delta T) \end{bmatrix}$$

$$H_{\Delta p} = \begin{bmatrix} 0 & 0 & 0 & \Delta T & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \Delta T & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \Delta T & 0 & 0 & 0 \end{bmatrix}$$

The two forms of measurements are applied under different cases. When GPS data is available, the position measurement, $\underline{Z}_p$, is used. However, when GPS is unavailable, the error states are propagated using the position rate change, $\underline{Z}_{\Delta p}$, calculated using the last two position estimates.

Even though the Kalman filter used is the same KF used to integrate the MAPS/GPS, the effectiveness of the KF when applied to the IMU/GPS is reduced since it uses nine states to estimate position, velocity and heading errors but does not consider accelerometer biases and gyroscope drifts. This will be shown clearly in the section describing System Performance.

A possible solution to aid the KF in better tracking the IMU states is to have a feedback loop sending position, velocity and attitude (PVA) data back to the IMU to reinitialize the states prior to calculating a navigation solution. Figure 5.27 compares the system diagrams for both the MAPS/GPS and the IMU/GPS system with feedback. It is clear that the MAPS does not rely on feedback of the position, velocity and attitude (PVA) states in calculating its navigation output. The KF is used to improve on the MAPS output by using GPS as its other measurement in reducing the error growth.

On the other hand, the KF sends PVA feedback to the IMU which it uses as its initial states when propagating its navigation solution. The improved output of the IMU is now used in conjunction with the GPS to get an even better estimate of the system PVA. This feedback control makes using lower-grade sensors such as the IMU possible. Otherwise, the rapid error growth would make the output unusable. Figure 5.28 shows general error curves for the

MAPS and IMU when operating with GPS and a KF. These trends will be characteristic for position, velocity and attitude errors.

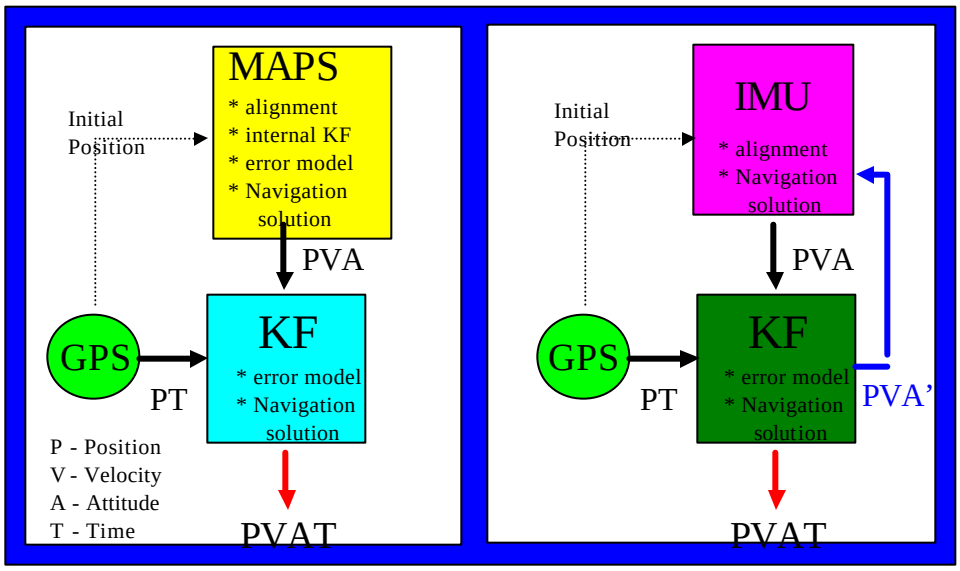


Figure 5.27: KF System Diagram.

In Figure 5.29, there is a distinct difference in the error growth curves for the IMU and MAPS when they are in stand-alone mode. As explained earlier, the main contributors to the disparity in performance are the gyroscope drifts and accelerometer biases, which are smaller for higher quality sensors such as those used by the MAPS. Aside from this, the MAPS uses its own internal Kalman Filter to provide error corrections when calculating position, velocity, and orientation through a built-in navigation solution. The integration of IMU with GPS through an external KF will greatly improve the accuracy of the navigation output, approaching that of the MAPS/GPS.

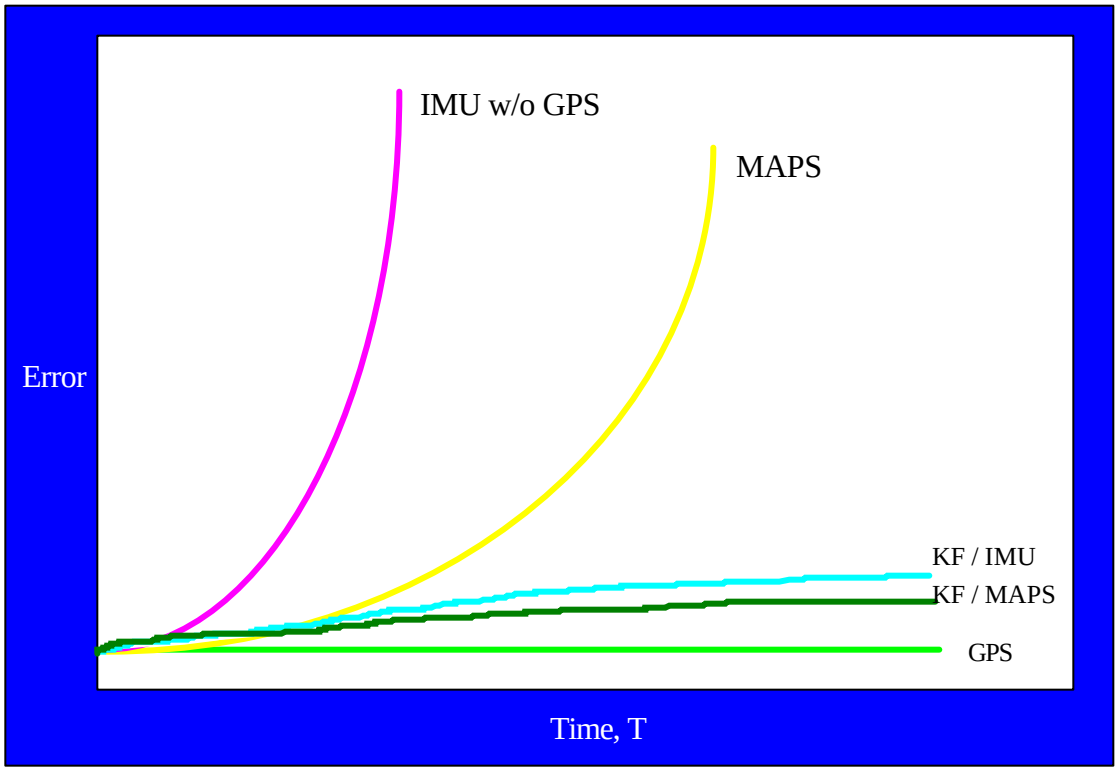


Figure 5.28: General Error Curves for MAPS/GPS and IMU/GPS.

In Figure 5.29, a more detailed description of the position error curve for the IMU/GPS is given. The graph clearly shows that the IMU position error is prevented from growing rapidly when aided with GPS position updates, received every second. The Kalman filter / IMU uses the current update to propagate position until the next update. The IMU / Navigation Solution (IMU/NAV) is constantly updated to the position correction of the KF. However, this error smoothing is only effective when GPS is available. When GPS is lost, the IMU/NAV position error eventually grows exponentially along with KF errors as shown in Figure 5.30.

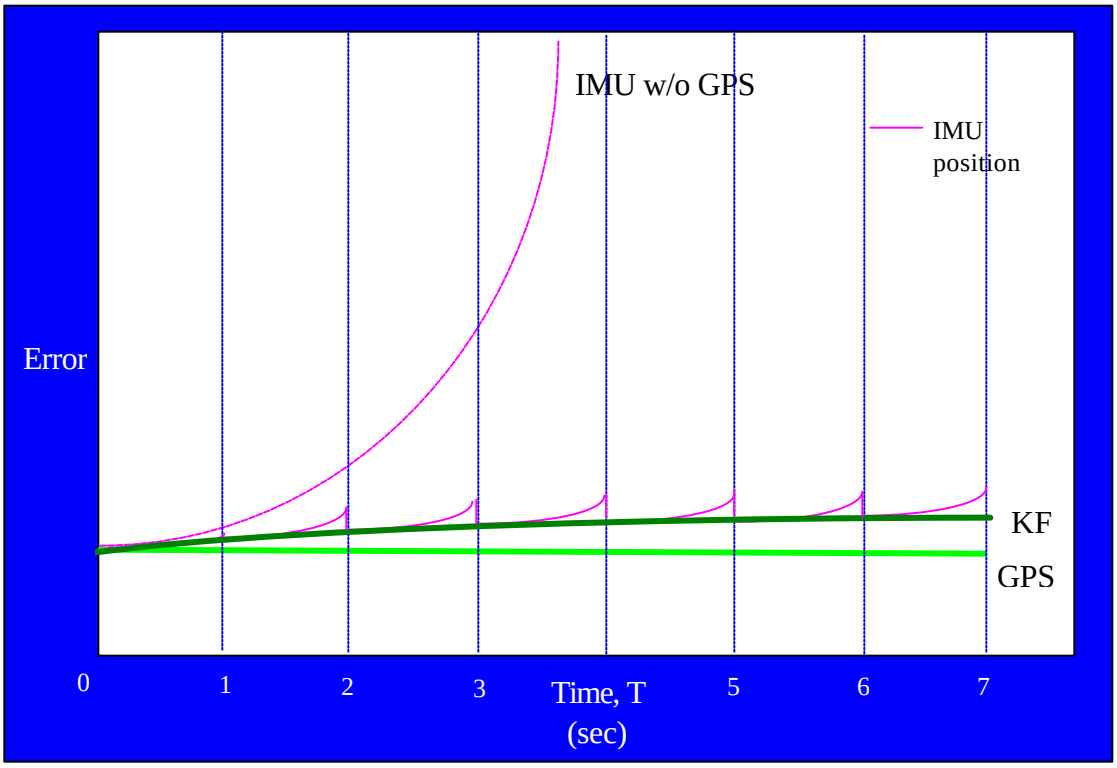


Figure 5.29: Detailed Error Curve for IMU/GPS.

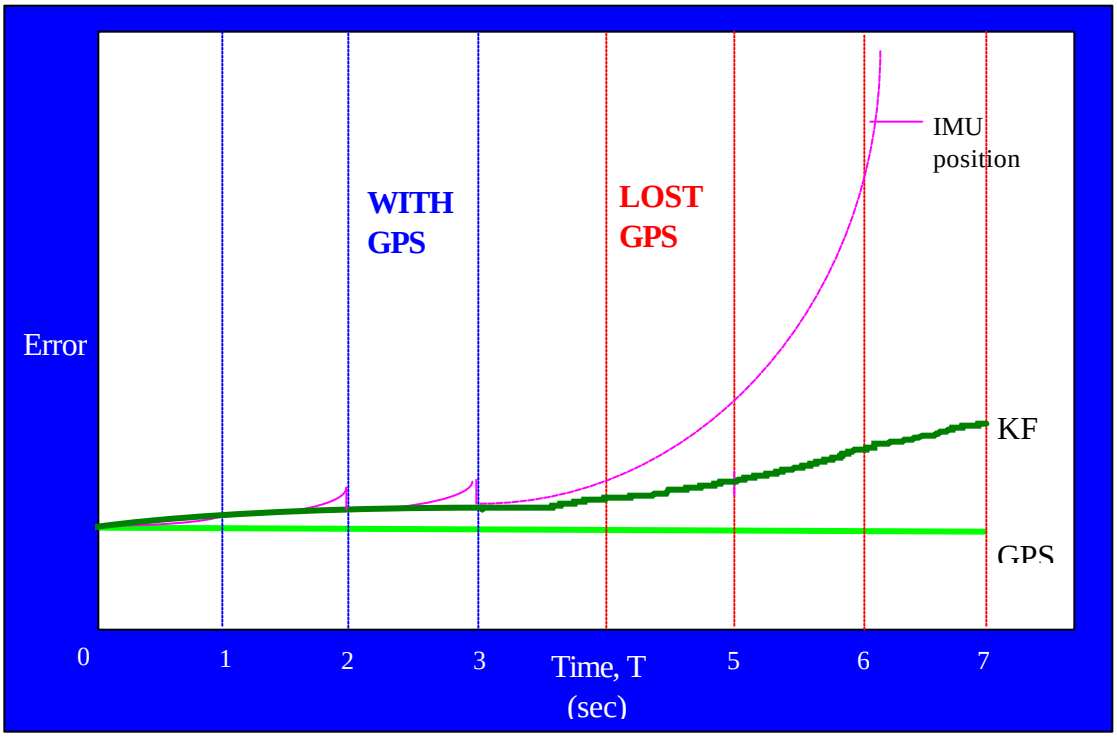


Figure 5.30: Error Curve during Loss of GPS.

Timing

As explained earlier in Chapter 3, the synchronization of the system components is essential to accurately use data at its valid time. Since only the GPS data is time-tagged, it will be used as the reference for synchronization.

The Main POS Processor's (MPP) system clock will keep track of the current time that it tags onto the IMU data. However, the PC clock has a drift. This is solved by sending a mark input pulse to the RT-20 GPS receiver's I/O port and requesting a position precisely time-tagged to the mark input resulting in a highly accurate reference time stamp. The time stamp is then used to reset the PC clock. This process is repeated every minute.

There are delays or lags associated with message transmission including line delays, output period delays (up to 0.08 sec), and processing delays. Just like in the timing of the MAPS, the IMU can be synchronized with GPS better by manually gathering position data from the output of the IMU and the Kalman Filter, and comparing them. By using a positional reference point, a time offset can be determined and used to adjust the time of the IMU data.

System Performance

In order to fully understand the significance of the results of testing the performance of the integrated system, the intermediate steps of IMU alignment and navigation will first be analyzed. Each step will be referenced to known values to accurately benchmark the individual processes. Also, the Kalman filter will first be tested then tuned using simulated GPS position. Several dynamic tests are also performed using the MAPS/GPS positioning system as the benchmark.

IMU Alignment

IMU Alignment consists of the coarse alignment stage and the fine alignment stage. Since the fine alignment algorithm has not been tuned properly, only the coarse alignment stage is tested. Once fully aligned, the Honeywell MAPS unit is used as the known reference orientation. The coarse alignment time and the true yaw are varied and the results are presented in Tables 5.25 and 5.26.

Table 5.25: IMU Coarse Alignment (Time Variant).

Time (sec)	Orientation Error (MAPS – IMU) MAPS Orientation (deg): Roll = 2.58, Pitch = -2.81, Yaw = 98.09					
	Roll Error (deg)		Pitch Error (deg)		Yaw Error (deg)	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
60	0.07	0.02	-0.027	0.013	-1.956	2.908
120	0.067	0.029	-0.01	0.006	1.967	0.835
180	0.067	0.028	-0.01	0.016	1.138	1.203
300	0.072	0.021	-0.011	0.02	1.345	1.024

Table 5.26: IMU Coarse Alignment (Yaw Variant).

Yaw (deg)	Orientation Error (MAPS – IMU) Coarse Alignment Time = 120 sec					
	Roll Error (deg)		Pitch Error (deg)		Yaw Error (deg)	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
0	-0.052	0.029	0.041	0.011	2.31	1.67
90	0.033	0.008	-0.069	0.032	-1.79	1.02
180	0.074	0.039	-0.021	0.006	1.96	0.77
270	-0.036	0.005	0.082	0.051	-1.42	0.53

As illustrated by the above results, the dependence of the orientation errors on the current yaw angle may be attributed to small errors in the calculation and application of the coefficients determined during manual calibration.

IMU Navigation

Testing of the output of the navigation solution of the IMU is performed using a static and dynamic test. In the static test, the IMU is allowed to propagate its position, velocity and orientation from a known initial position and orientation. In the dynamic test, the IMU is moved around a predetermined path in a short amount of time. Once the IMU position errors grow large, navigation becomes difficult.

Static Test

The known position used is referenced to the CIMAR laboratory (Latitude = 29.646418 deg., Longitude = -82.349352, Altitude = 12.34 m.). The known orientation is obtained from the initial alignment performed for 10 minutes. Static data is collected for 60 minutes. The difference between the known position values and the navigation output are plotted in northing error, easting error, altitude error, and radial position (2D) error ($\sqrt{(North\ Error)^2 + (East\ \ Error)^2}$) against time in Figures 5.31, 5.32, 5.33 and 5.34.

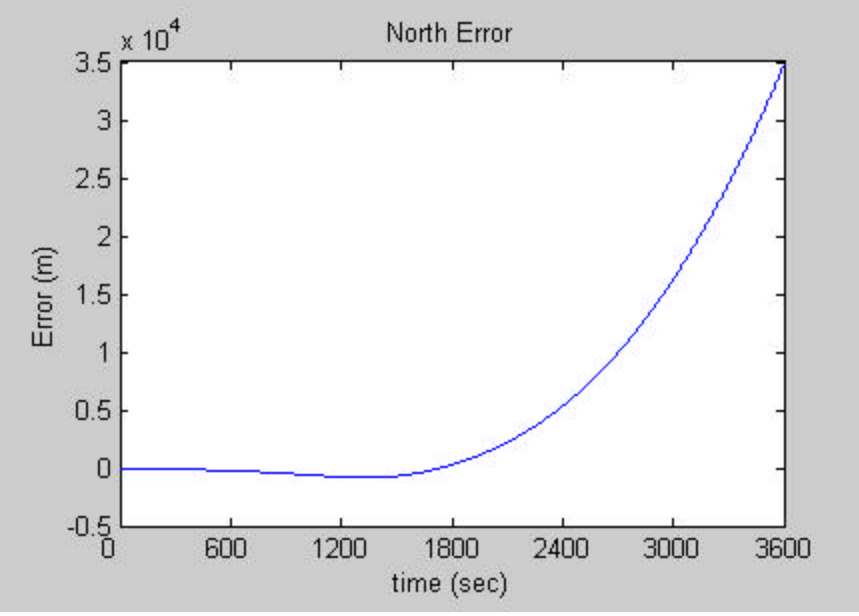


Figure 5.31: IMU Static Test Northing Error.

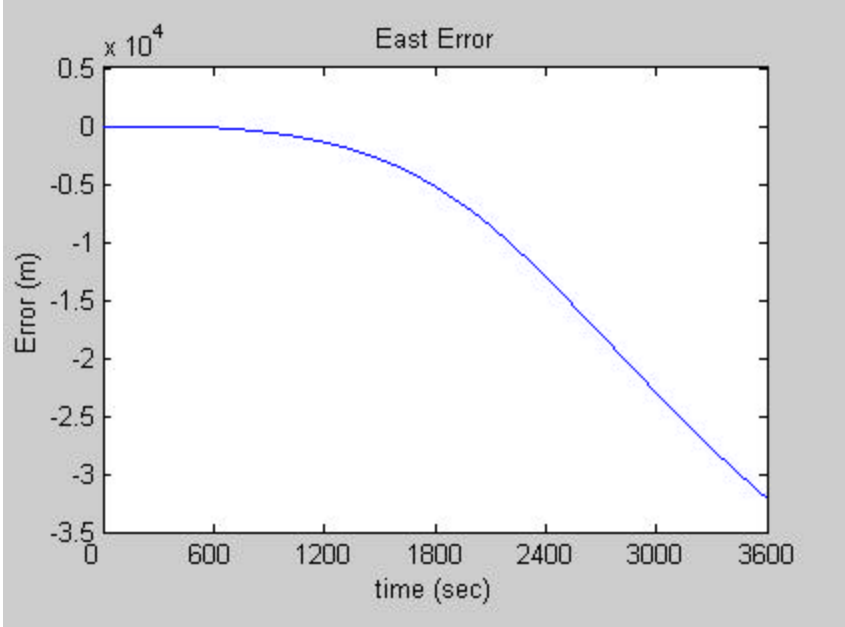


Figure 5.32: IMU Static Test Easting Error.

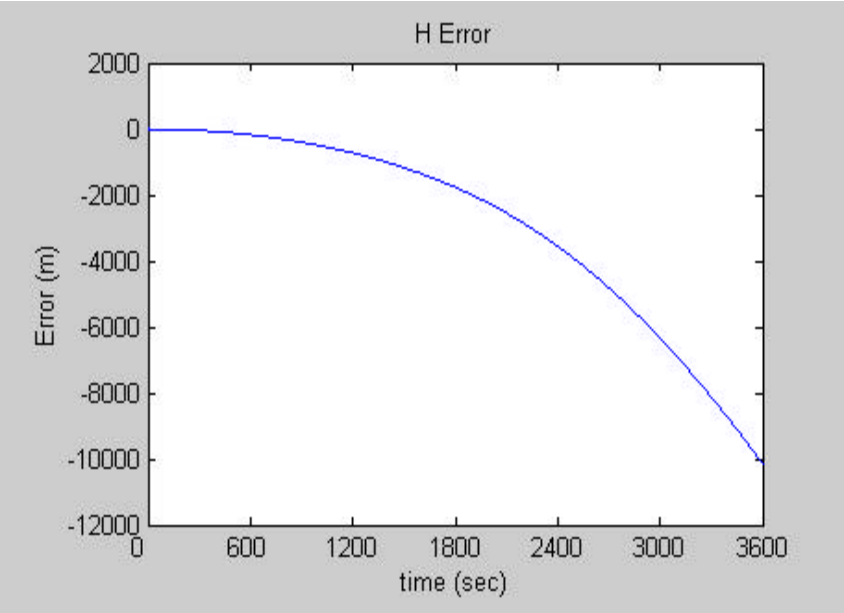


Figure 5.33: IMU Static Test Altitude Error.

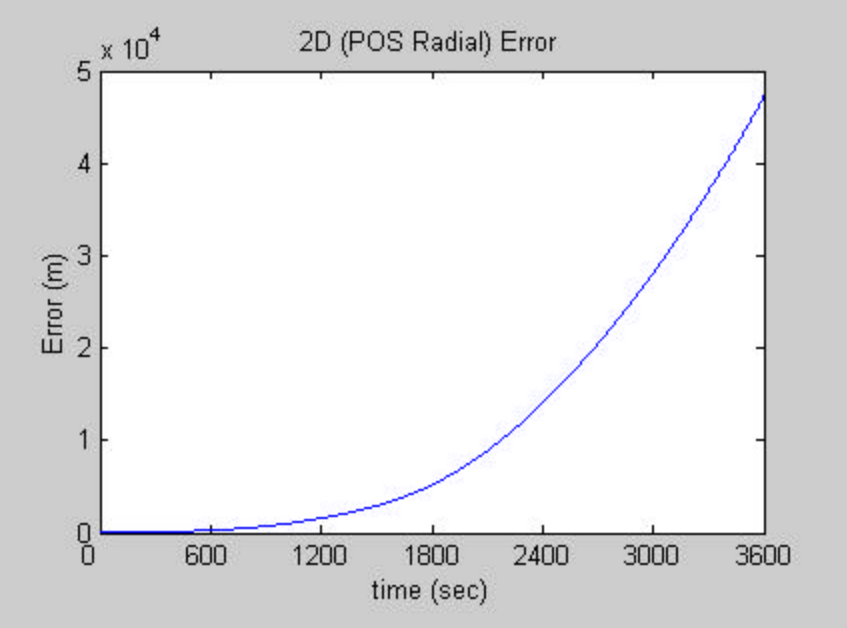


Figure 5.34: IMU Static Test Pos Radial Error.

The orientation errors are shown in Figures 5.35, 5.36, and 5.37.

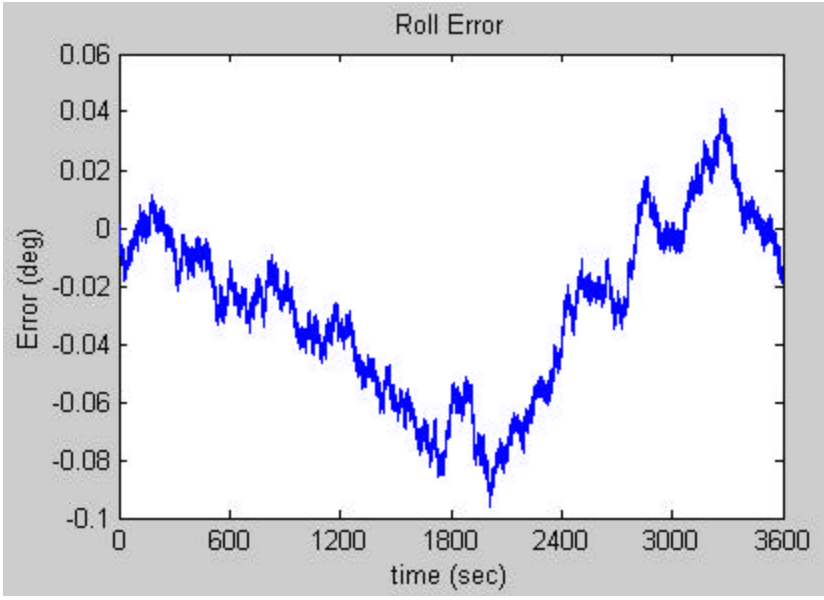


Figure 5.35: IMU Static Test Roll Error.

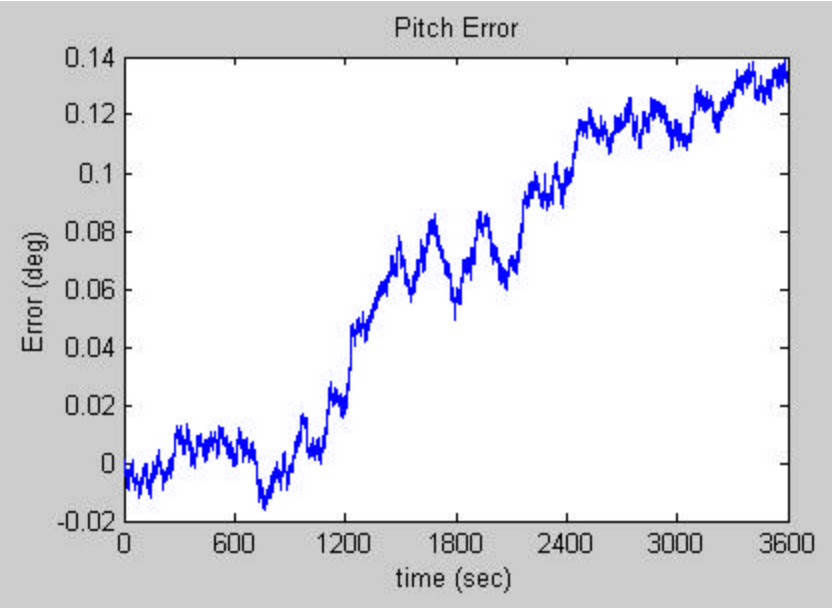


Figure 5.36: IMU Static Test Pitch Error.

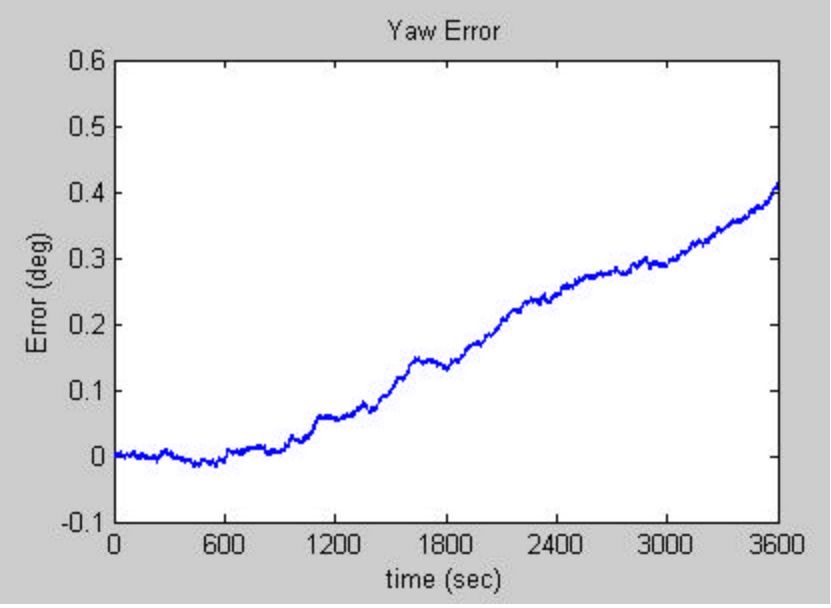


Figure 5.37: IMU Static Test Yaw Error.

The results of the static test for both position and orientation errors fall within the maximum errors calculated when using the performance specifications of the IMU for accelerometer biases and gyroscopes drifts. The IMU is supposed to deliver a single-axis

position accuracy to within 70 kilometers after 60 minutes. Single axis angular drift is rated to be no more than 1.0 degree per hour.

Dynamic Test

A simple way to test the dynamic output of the navigation solution of the IMU is to move the sensor unit along a known path for a period of no more 40 seconds. This ensures that the 2D position drift will stay below 10 meters. The actual test shows that after 25 seconds, the IMU position drifts 0.8 meters. Also, the position drift is magnified with motion in the same direction as the apparent bias. Here, the northeast becomes obvious even when the IMU is stationary like at the end of the run.

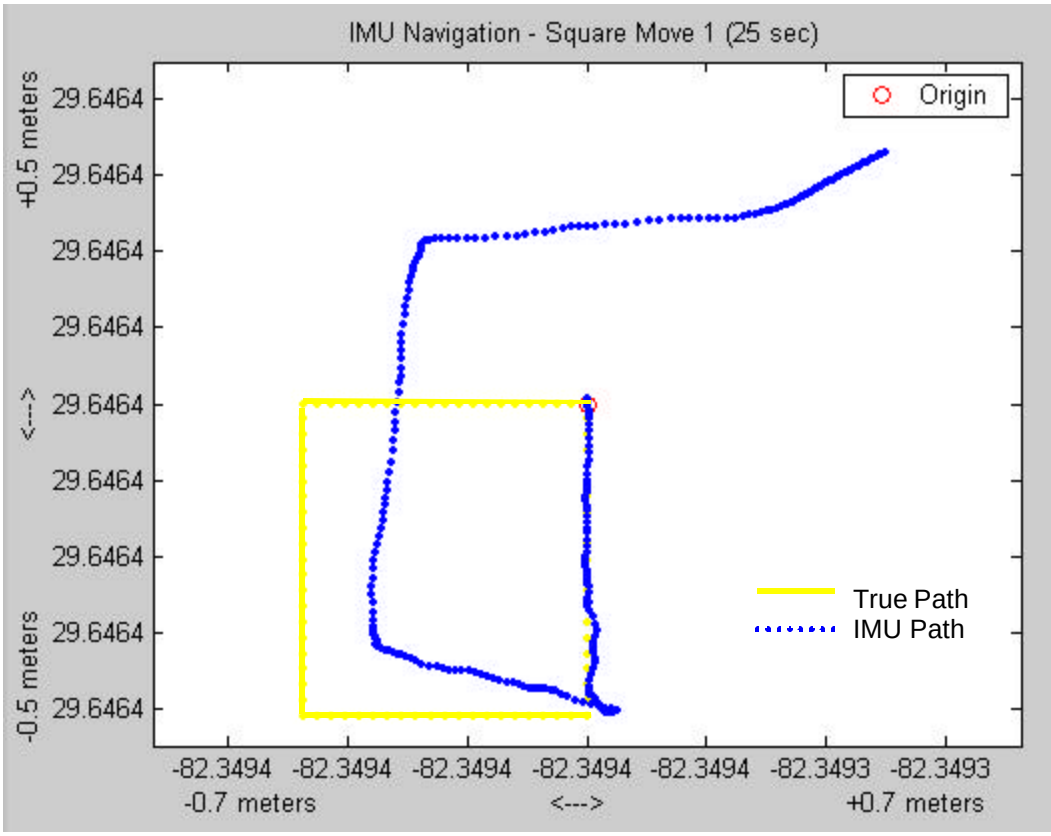


Figure 5.38: IMU Dynamic Test 1.

A second dynamic test is performed while the IMU is on the navigation test vehicle (NTV) while it traverses 30 meters in the west direction. Over a span of 40 seconds, the IMU travels 37 meters in the west direction with a north drift of 3 meters. This is shown by Figure 5.39. Figure 5.40 illustrates a third test wherein the NTV makes a counterclockwise loop to end up near its start position. Here, the IMU output position shows an 8 meter 2D error with a predominant southwards drift.

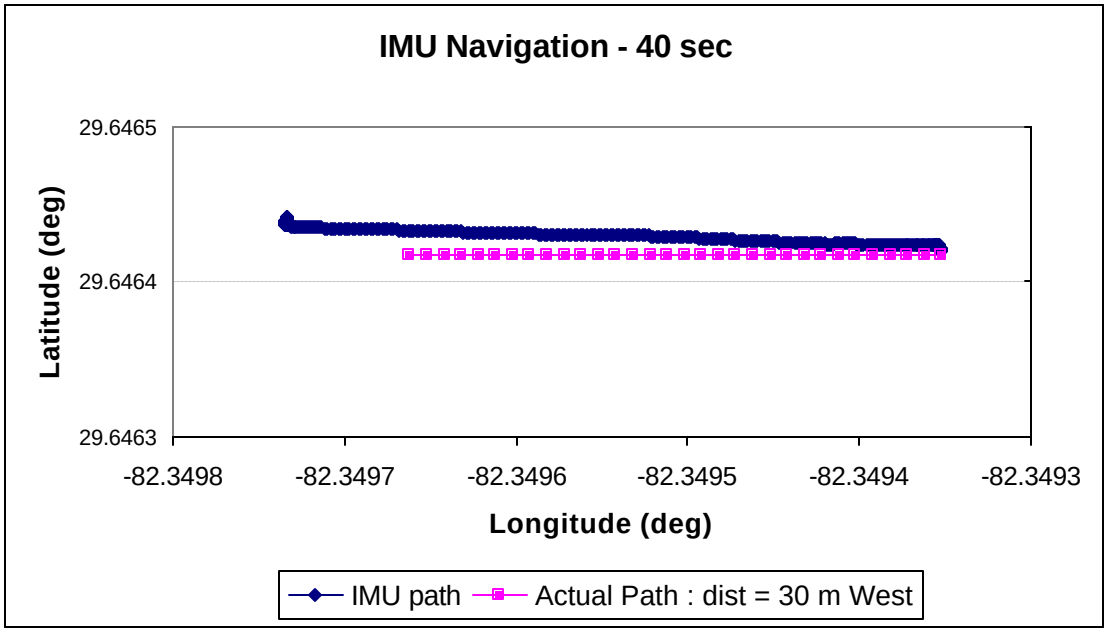


Figure 5.39: IMU Dynamic Test 2.

Using Simulated GPS Position

The first step is to simulate GPS position. This will allow better initial tuning of the Kalman Filter (KF).

Under Ideal Conditions

The optimal performance of the integrated IMU/GPS system is measured by using a set-up that simulates ideal conditions. In this case, ideal conditions refer to the presence of good GPS position updates (at 1 hertz) at all times and with the IMU

accurately aligned. There are three tests run to measure system performance under ideal conditions.

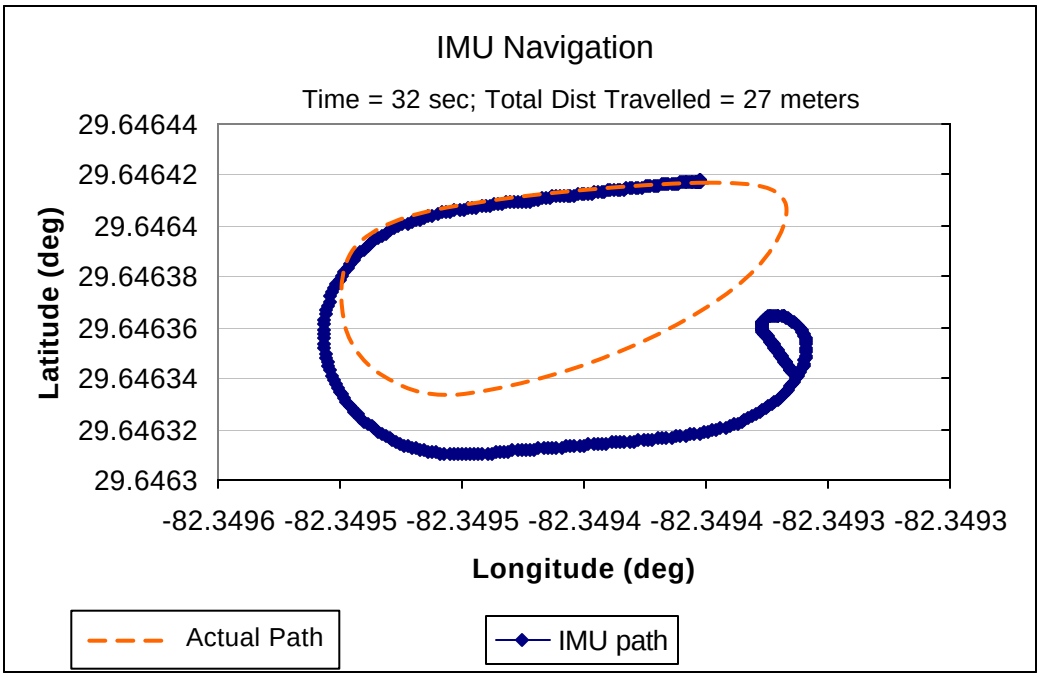


Figure 5.40: IMU Dynamic Test 3.

Static Test Using Reference GPS Position

By setting the GPS position to a fixed value previously surveyed, the Kalman Filter (KF) takes advantage of tracking a constant position measurement, in the reference GPS position, greatly enhancing its performance. The KF is then tuned by directly measuring the filter position versus the constant GPS position and minimizing the errors between them.

Once again the reference GPS position used is that of the CIMAR laboratory. After the filter parameters are adjusted to that of the IMU, a static test reveals that the filter is able to maintain a position accuracy of 0.5 m to that of the GPS position within 600 seconds (or 10 minutes). Beyond this point, the IMU error growth is so rapid (error

> 1200 m) that the filter is not able to keep up, resulting in large error oscillations. Figure 5.41 clearly illustrates this.

Static Test Using Tuned Kalman Filter

Further tuning of the Kalman filter is performed by adjusting the Q matrix that contains the error rates for each state. Initially, Q is based on the parameters that are read in through the filter configuration file (see Appendix C). By multiplying the tilt rates by a factor of 10, better filter convergence and error tracking is achieved. In Figure 5.42, the filter is able to track the GPS position to within 0.3 m over a span of 1000 seconds.

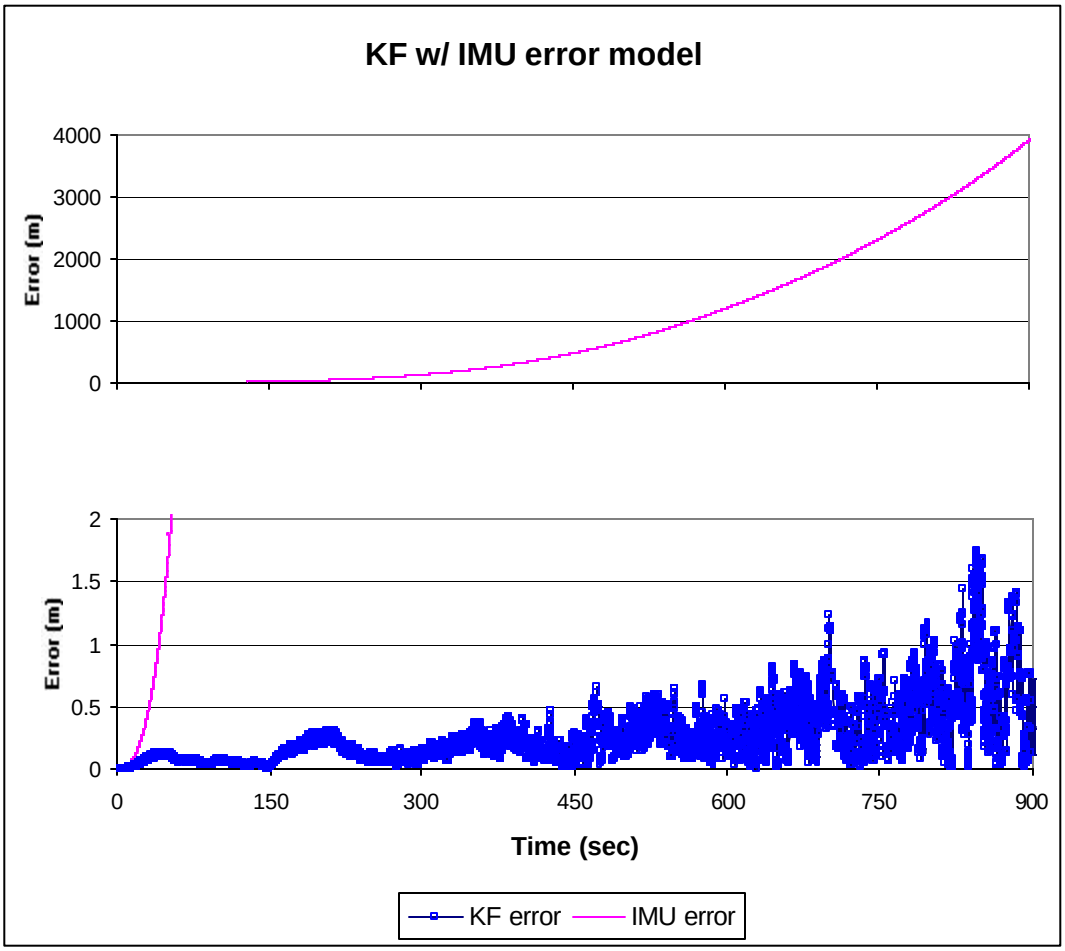


Figure 5.41: KF w/ IMU error model.

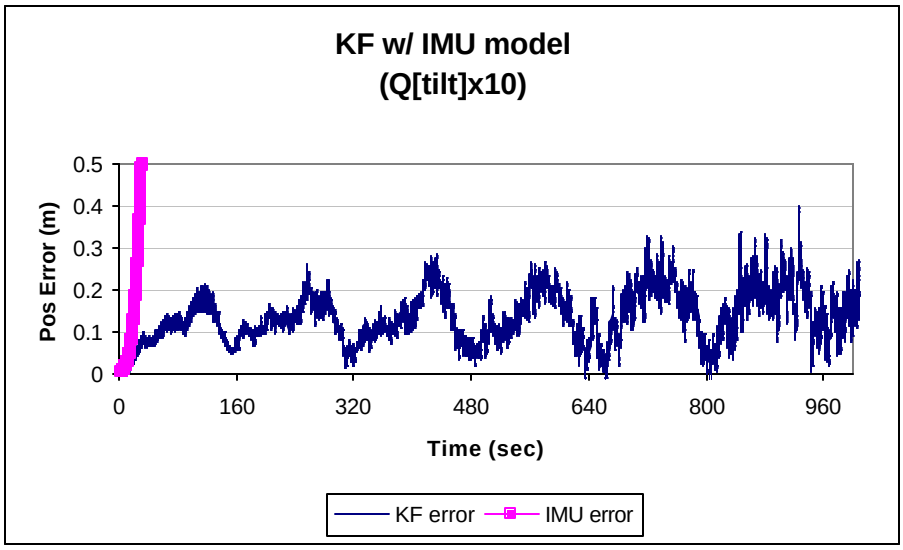


Figure 5.42: Tuned KF w/ Q = 10 x [tilt rates].

Static Test Using KF Updates To Reset IMU Navigation

Even with the KF tuned, the IMU position error growth becomes increasingly significant over time. The MAPS can navigate for hours before the position error reaches the kilometer range. It is impossible for the Kalman filter to output a good estimate since the IMU errors are orders of magnitude greater than the desired submeter accuracy. In order to effectively use the IMU for extended periods of time, a navigation reset is performed. This navigation reset requires position, velocity and orientation updates from the Kalman filter to the IMU. As explained earlier, a feedback loop now exists between the KF and the IMU. The time span between updates is determined by two factors: (1) time when the KF errors have converged to a minimum, and (2) time when the IMU 2D position error does not exceed 500 meters. The key is to let the KF run as long as possible before having to reset the IMU. Based on the previous static tests, a reset time

span of 320 seconds is selected. Figure 5.43 shows that the errors stay below 0.25 m over a time span of 1800 seconds (30 minutes).

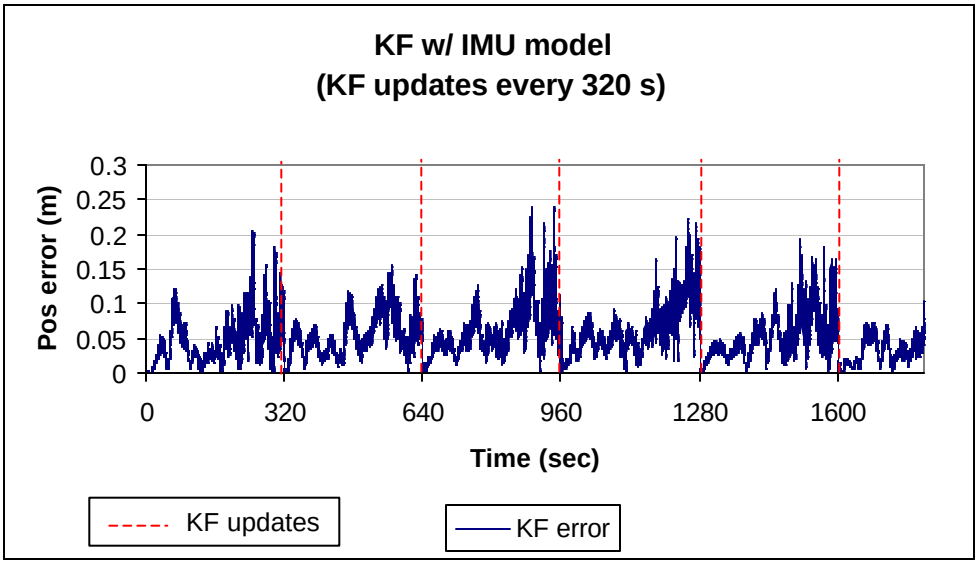


Figure 5.43: KF & IMU with KF updates every 320 sec.

It should be noted that the orientation plays a major role in the performance of Kalman filter. Steady gyroscope output is essential in allowing the filter to effectively model the position errors. To prove this point, a final static test is performed with the orientation set to a fixed value (forcing no drifts). In Figure 5.44, it is obvious that the KF output is greatly improved as the position errors stayed below 0.14 m for 1500 sec.

Under Adverse Conditions

The main purpose of the Kalman Filter is to provide smoothened position, velocity and orientation output even in the event of GPS loss. Since the performance of the filter has already been investigated under ideal static conditions, using the same set-up, the next step is to simulate temporary GPS loss. The behavior of the filter during GPS loss is studied giving emphasis to duration and frequency of GPS loss.

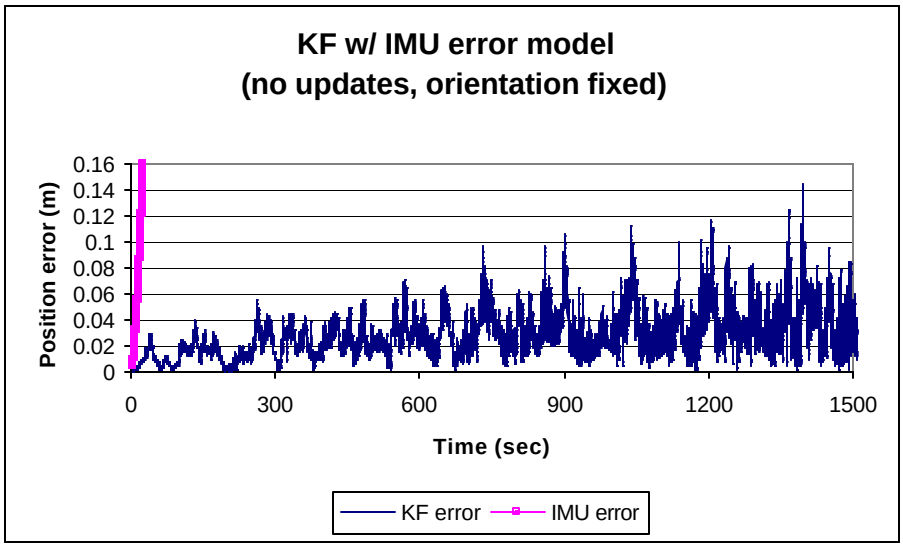


Figure 5.44: KF & IMU with orientation fixed.

Temporary GPS Loss

Figures 5.45, 5.46, and 5.47 show the KF error curves during a single GPS loss occurring at 200 sec. for durations of 10 sec., 30 sec., and 60 sec. respectively. The filter manages to keep errors down only during the 10 second outage. However, it can be seen that once GPS lock is reacquired, the position errors drop significantly and within 20 seconds error converges to a minimum.

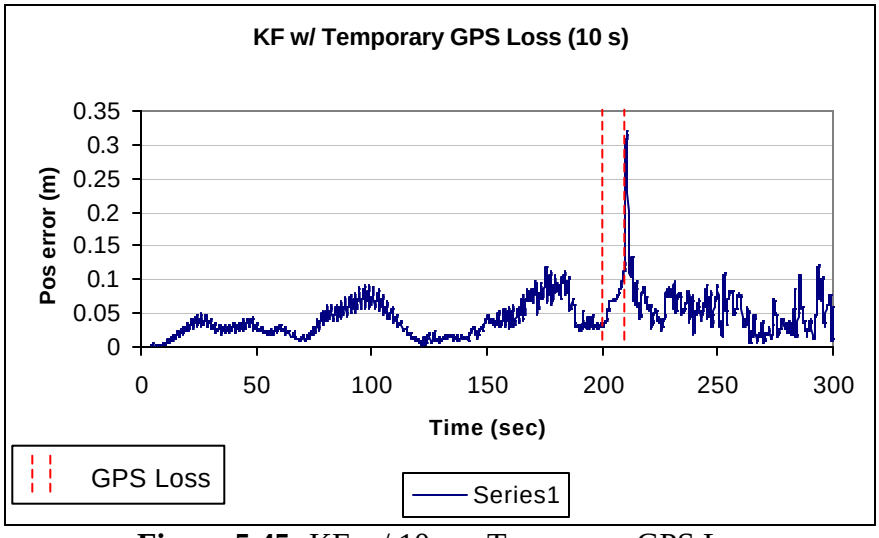


Figure 5.45: KF w/ 10 sec Temporary GPS Loss.

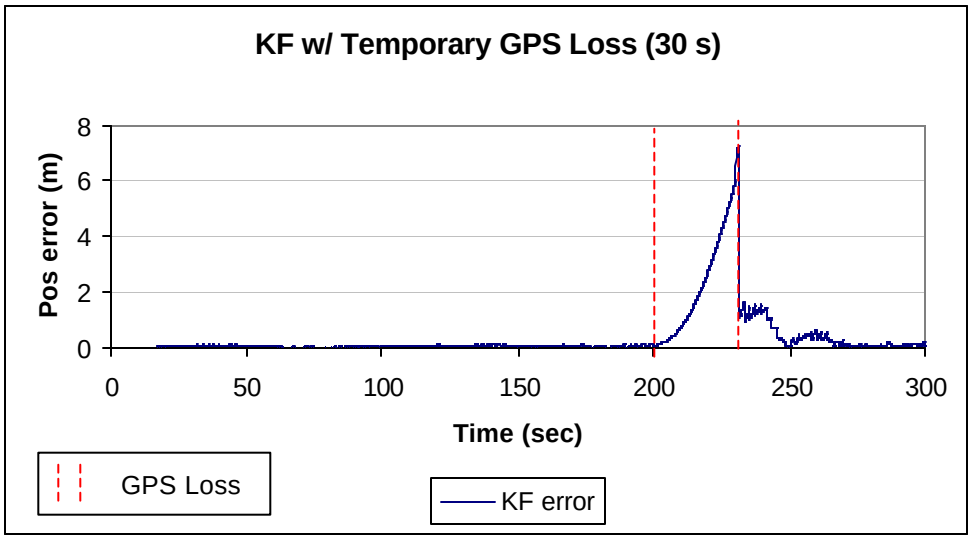


Figure 5.46: KF w/ 30 sec Temporary GPS Loss.

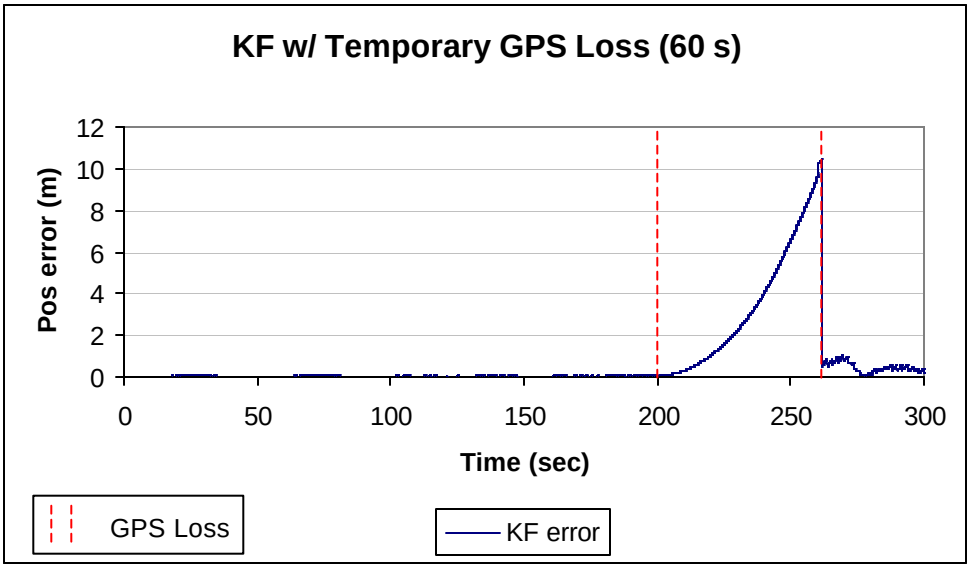


Figure 5.47: KF w/ 60 sec Temporary GPS Loss.

The ability of the Kalman filter to recover after several short losses of GPS is also tested. Using 10 second GPS outages occurring every 150 seconds, the filter shows it can good position accuracy only for the first two outages (see Figure 5.48). The third GPS outage results in a 2 meter error jump which is a result of the higher IMU errors.

In comparison, the same GPS loss sequence is tested with the KF updating the IMU navigation after 300 seconds. This resets the error growth and allows the filter to provide better estimates even during the third GPS loss. In Figure 5.49, the position error spikes reach no higher than 0.7 m. throughout the 600 second time span.

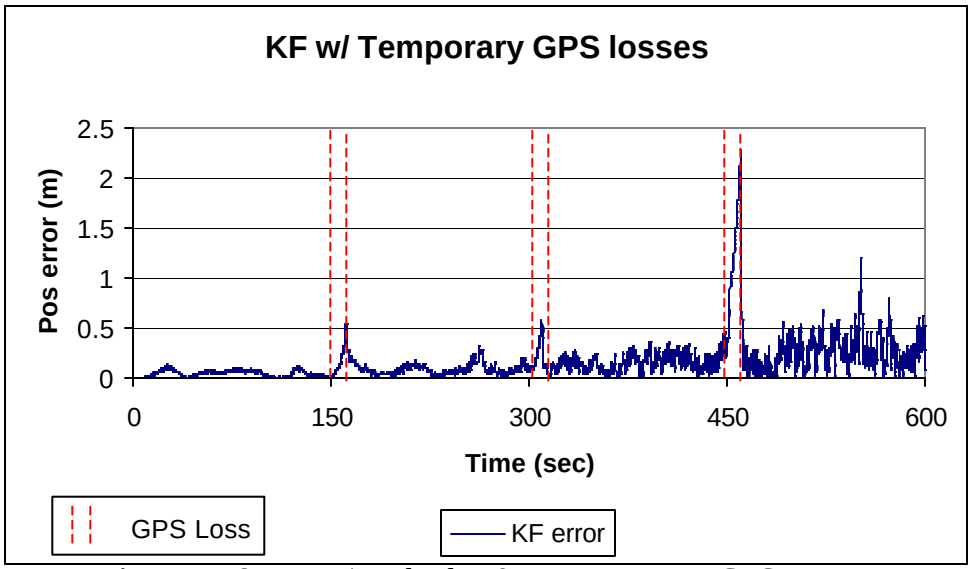


Figure 5.48: KF w/ Multiple 10 sec Temporary GPS Losses.

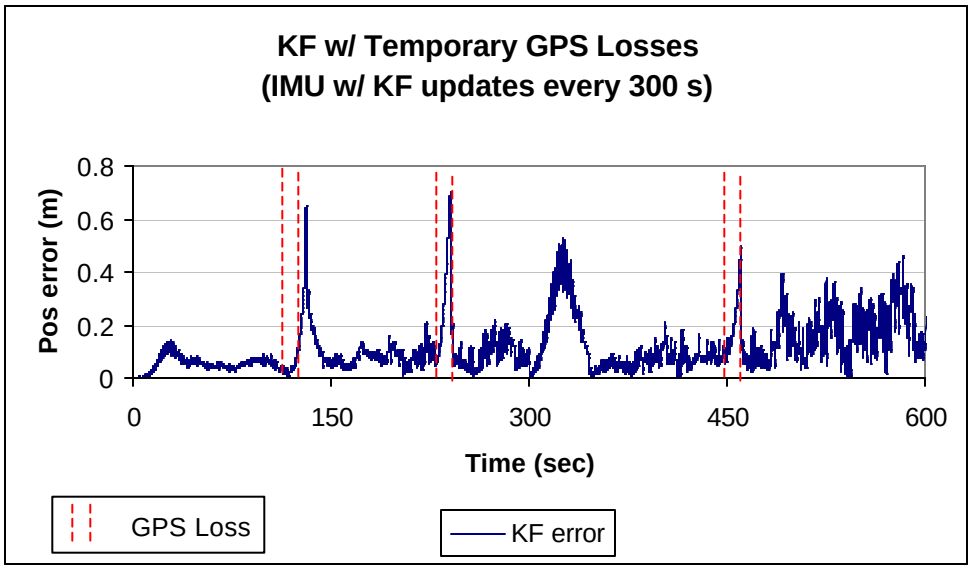


Figure 5.49: KF w/ Multiple 10 sec Temporary GPS Losses
IMU Updated By KF at 300 sec.

Using Actual GPS Position

With the KF tuned, the IMU/GPS is subjected to dynamic tests using actual GPS position data. All the data runs are conducted on Test Range 3 of the Air Force Research Lab in Tyndall Air Force Base using the Autonomous Mobility Research and Development System (AMRADS) as the vehicle platform. The benchmark being used is the MAPS/GPS system.

Dynamic Tests

With the availability of actual GPS position data at 1 Hz, the next step is to investigate the performance of the Kalman Filter (KF) as it tracks the position, velocity and attitude during dynamic or moving conditions. With the AMRADS using the MAPS/GPS as its main positioning system, the vehicle is manually driven given different conditions: pure translation, pure rotation, and combined translation and rotation. Also, as it will be shown later, the IMU/GPS performs adequately only for short periods of time. The data runs are thus confined to less than 200 sec.

Pure Translation

The objective of running under pure translation is to gauge the performance of the accelerometers by decoupling the effect of rotation as measured by the gyroscopes. In the test, however, the vehicle is driven manually which results in slight orientation changes. In Figure 5.50, it is clear how the position drift of IMU causes the IMU/GPS KF position output to diverge slowly away from the output of the MAPS/GPS. After 80 seconds, the total position RMS error of 1.09 meters, after a total travel of 48.56 meters eastward. Figure 5.51 shows the individual position errors in Local Geodetic Vertical coordinates (North, East, Down).

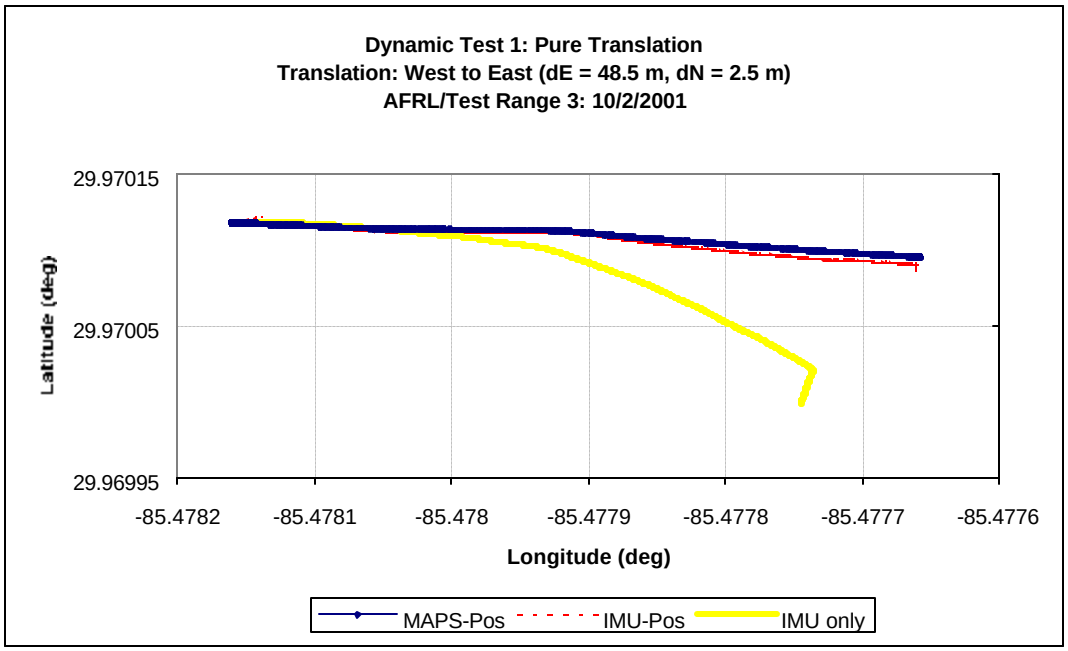


Figure 5.50: Dynamic Test 1 - AMRADS under Pure Translation

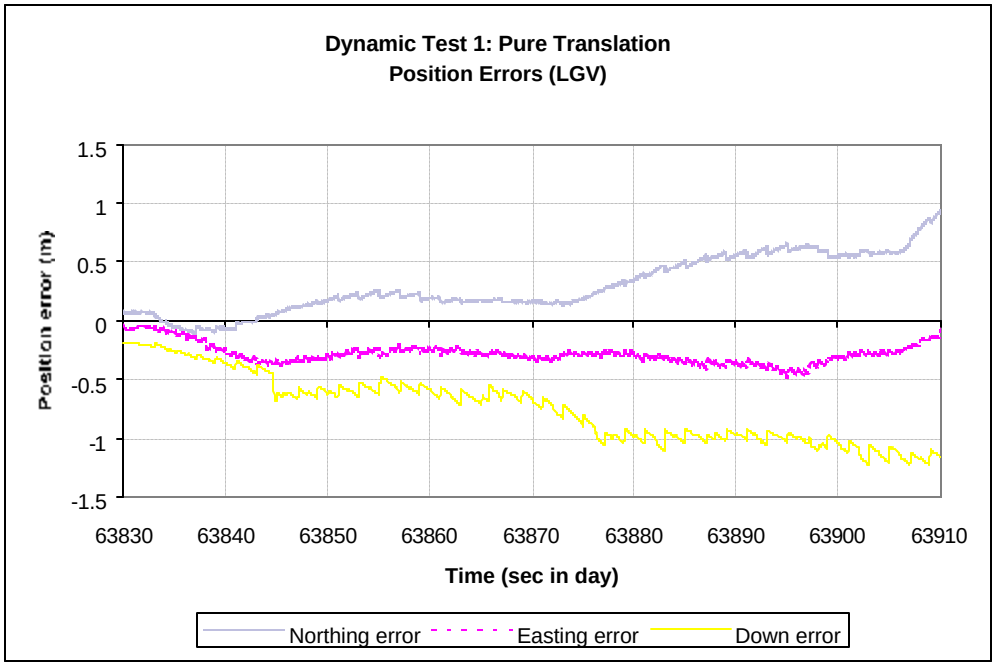


Figure 5.51: Dynamic Test 1 - LGV Position Errors

The apparent drift in the north is caused by a divergence of the IMU north velocity from the MAPS-Pos (see Figure 5.52), while the east velocities are closer (see Figure 5.53).

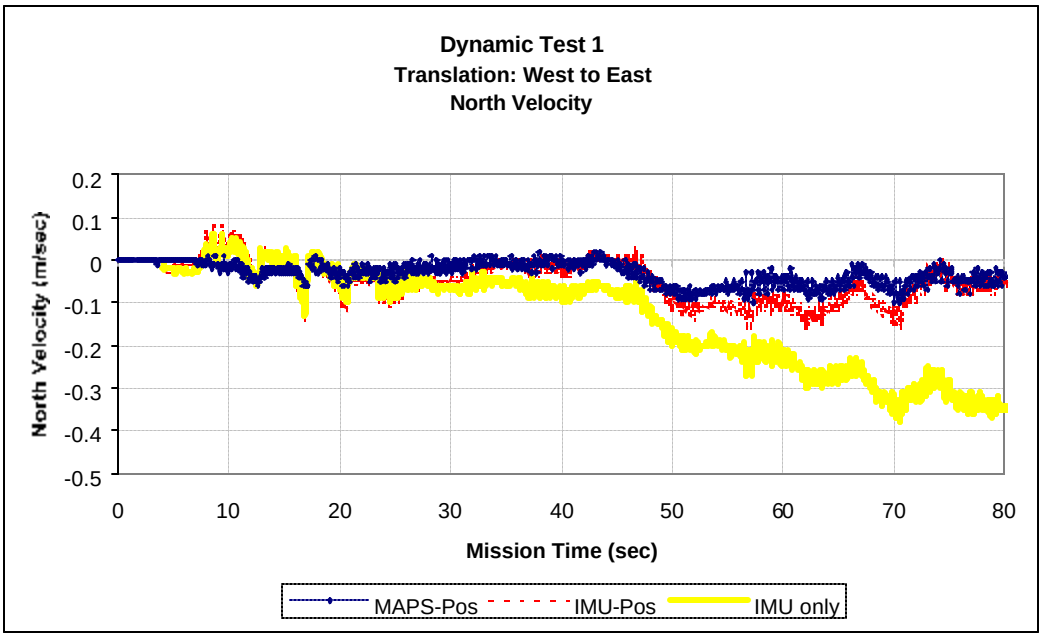


Figure 5.52: Dynamic Test 1 – LGV North Velocity

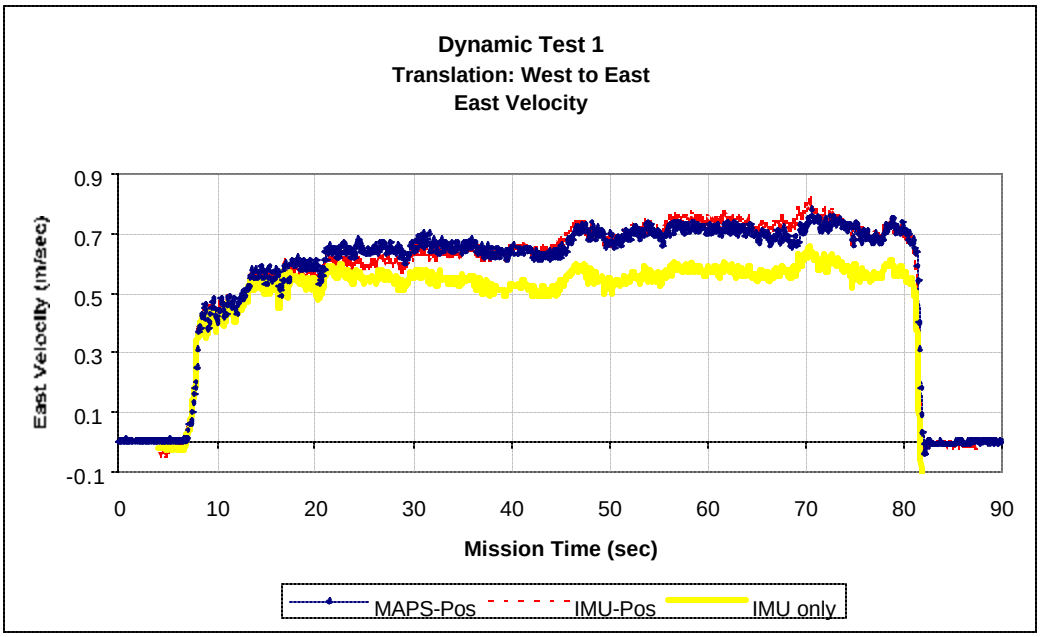


Figure 5.53: Dynamic Test 1 – LGV East Velocity

Throughout the entire run, while the KF tries to track the velocity, mean velocity errors of 0.05-0.1 m/sec in all directions results in eventual positional drift (see Figure 5.54).

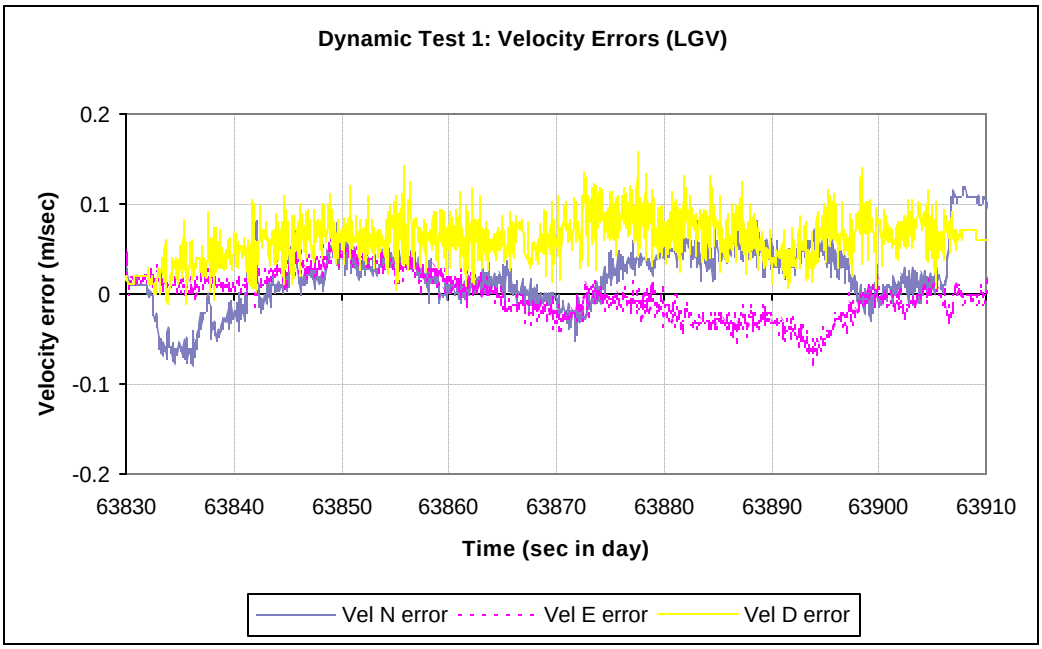


Figure 5.54: Dynamic Test 1 – LGV Velocity Errors

Figures 5.55, 5.56 and 5.57 show Roll, Pitch and Yaw for both IMU/GPS and MAPS/GPS. The IMU/GPS orientation curves are almost identical to those of the MAPS/GPS proving that the gyroscope output is good and the propagation algorithm of the angles is valid.

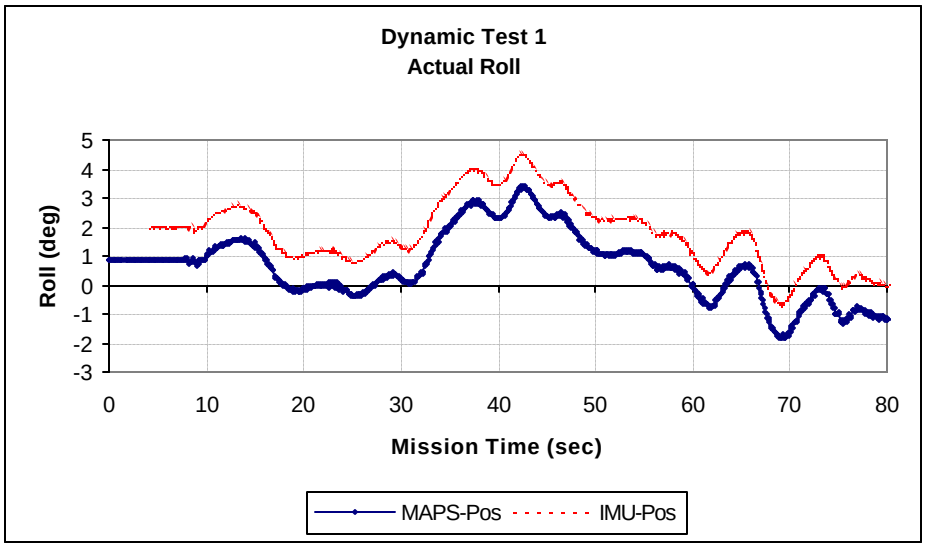


Figure 5.55: Dynamic Test 1 – Roll

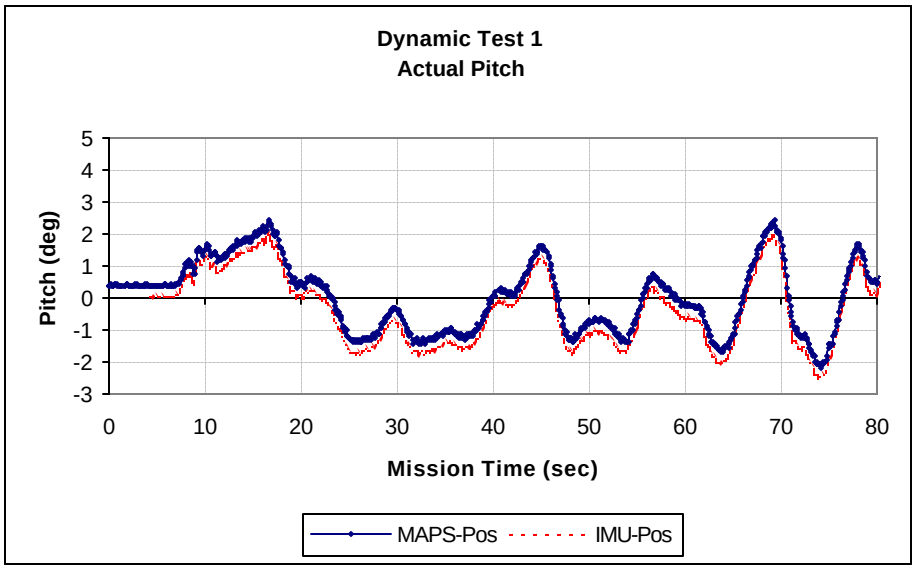


Figure 5.56: Dynamic Test 1 – Pitch

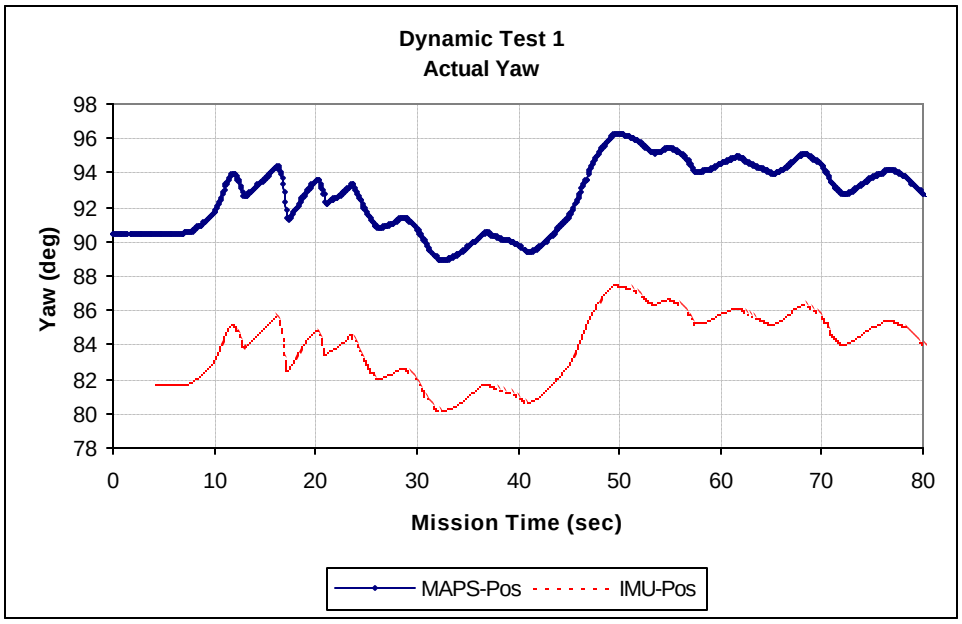


Figure 5.57: Dynamic Test 1 – Yaw

Finally, Figure 5.58 summarizes the orientation errors after each angle has been adjusted to eliminate the error offset caused by differences in initial alignment. All angles display errors ranging between ± 0.1 degree.

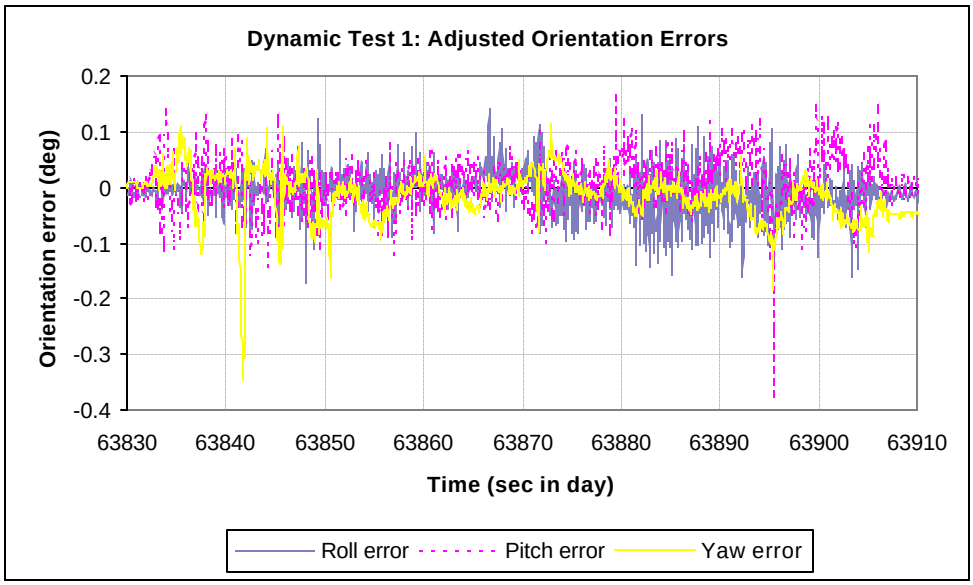


Figure 5.57: Dynamic Test 1 – Orientation Errors

Pure Rotation

To achieve close to pure rotation, the AMRADS is spun about its center counterclockwise for one complete revolution. The purpose of this test is to evaluate gyroscope performance under high dynamic conditions. However, since both the MAPS and the IMU are both located away from the geometric center of the vehicle, spinning the vehicle about its vertical axis also contributes linear accelerations to both inertial sensors. These accelerations are a function of the linear distance between the MAPS or IMU center and the center of the spin axis of the vehicle. The introduction of offset errors and high-speed rotation result in unwanted position drift and poor KF response. This is shown in Figure 5.58 for the MAPS/GPS system. The IMU/GPS system output shows even greater drift.

In Figure 5.59, the output orientation angle of interest, yaw, behaves identically for both positioning systems over one complete rotation (360 deg). Also, Figures 5.60 and

5.61 show that the roll and pitch curves are almost dead on except for offset values that are attributed to the MAPS and the IMU not being mounted on the same shelf. The individual orientation angle errors are given in Figure 5.62.

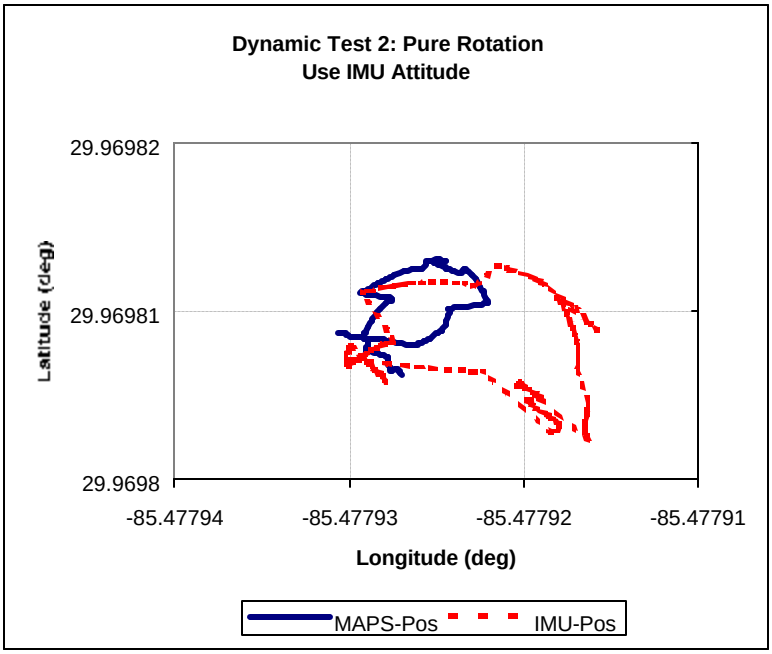


Figure 5.58: Dynamic Test 2 – Pure Rotation (ccw)

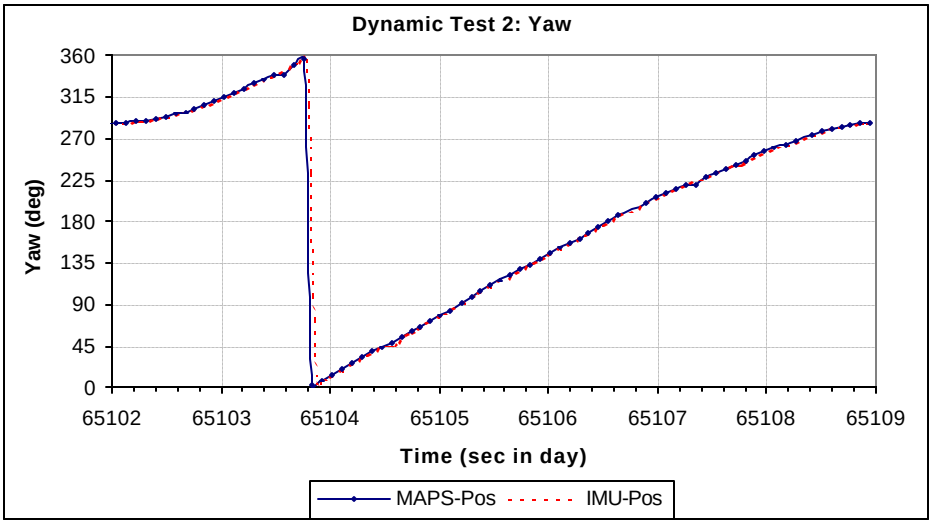


Figure 5.59: Dynamic Test 2 – Yaw

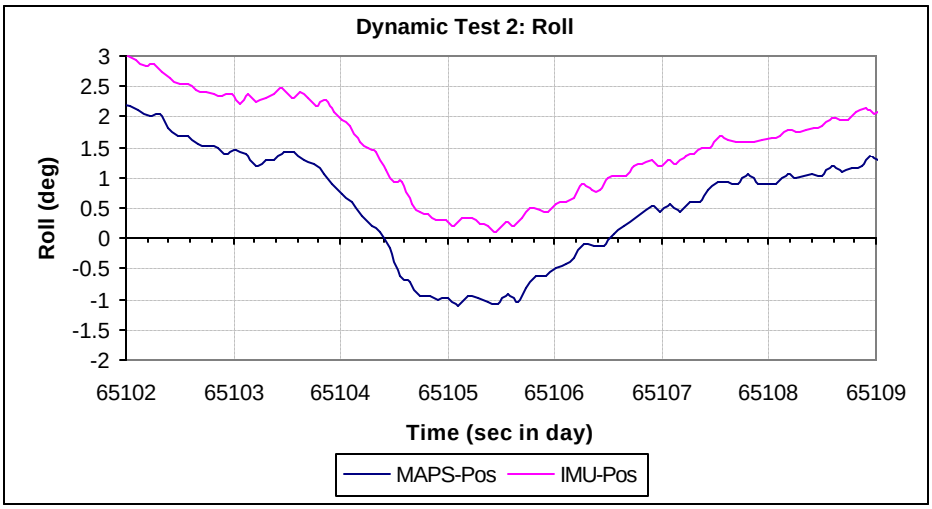


Figure 5.60: Dynamic Test 2 – Roll

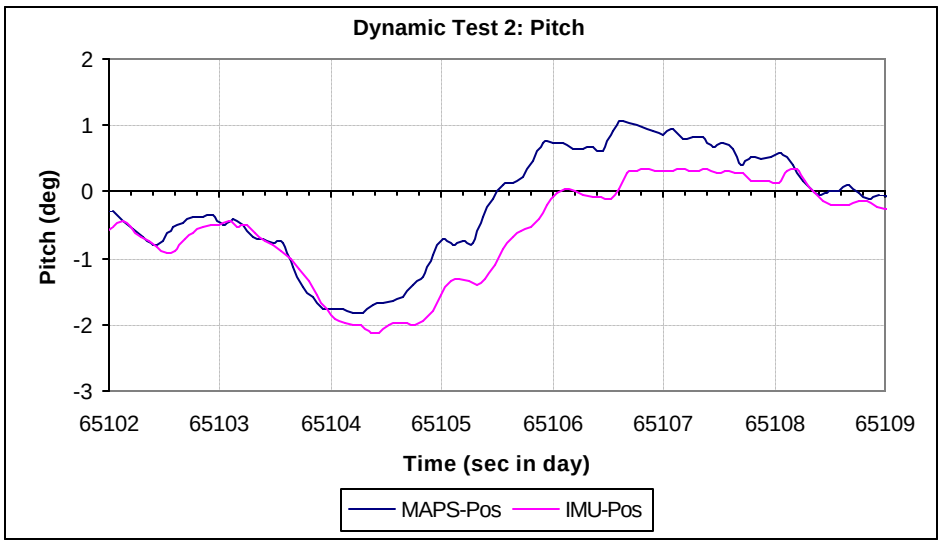


Figure 5.61: Dynamic Test 2 – Pitch

Again, the pure rotation test proves that the gyroscope outputs are quite adequate for calculating roll, pitch and yaw under dynamic conditions. However, it should be noted that this test is conducted for a very short period of time and that the large gyroscope drifts will eventually be a factor over time. In dynamic conditions, these gyroscope drift values are insignificant compared to the large angular rotations measured.

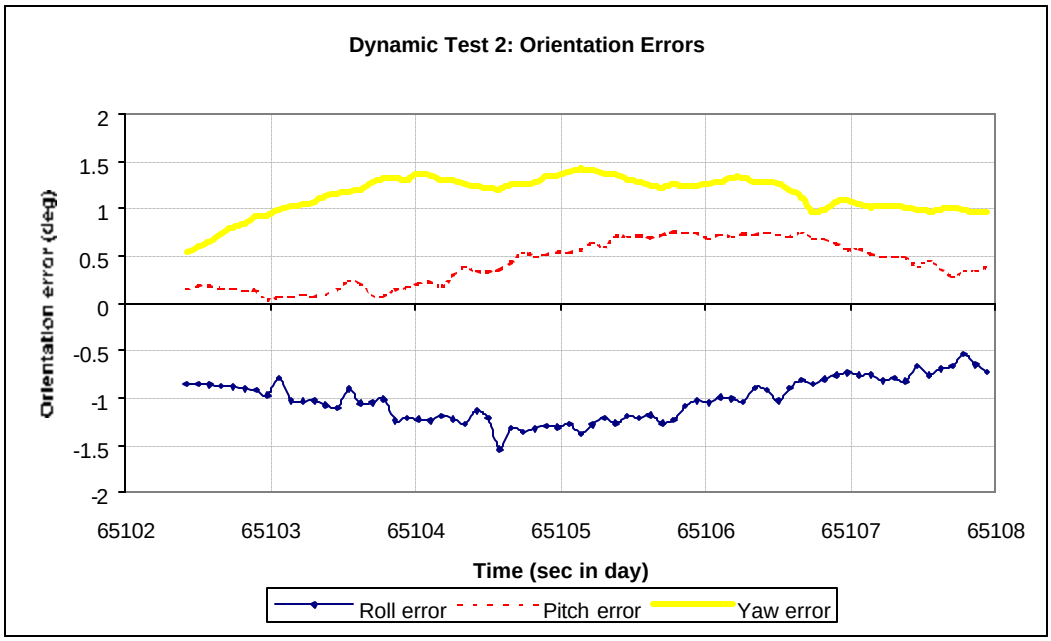


Figure 5.62: Dynamic Test 2 – Orientation Errors

Combined Translation and Rotation

In most situations of actual navigation, vehicle movement is characterized by a combination of both translation and rotation. Here, both gyroscopes and accelerometers play heavy roles in determining actual position, velocity and orientation of the vehicle.

CW Loop

Two different tests are performed. The first dynamic test (Dynamic Test 3) involves running the vehicle in a clockwise loop of approximately twenty meters in diameter. In this run, the velocity in the vehicle X direction is relatively constant.

In Figure 5.63, the position output of the IMU-Pos KF drifts slightly downward and recovers slightly at the end. The downward drift is a direct result of the IMU position rapidly drifting south and moderately to the east. The altitude is the least accurate position field and this is reflected in Figure 5.64.

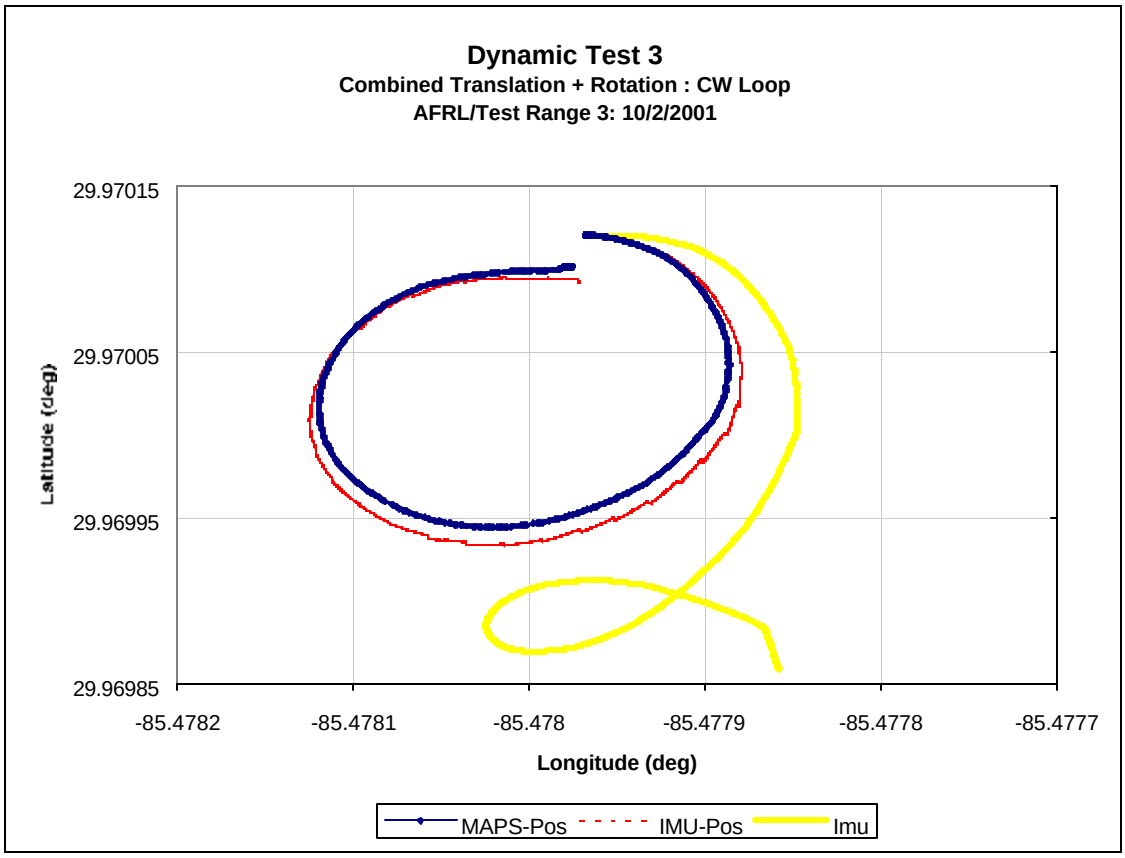


Figure 5.63: Dynamic Test 3 – Combined Translation and Rotation

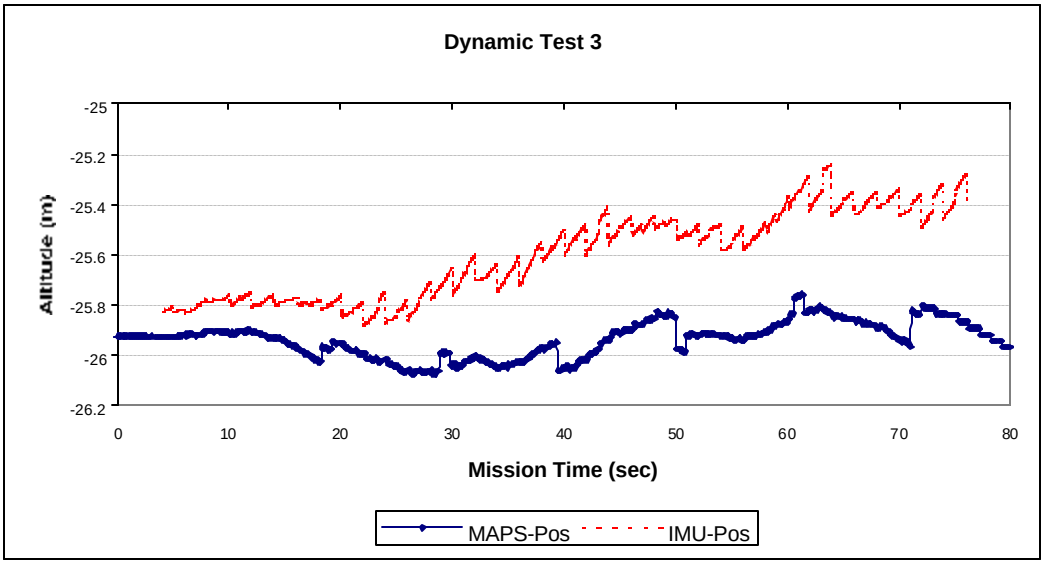


Figure 5.64: Dynamic Test 3 – Altitude vs. Time

As with the previous dynamic tests, the orientation angles prove to be consistent in both positioning systems. This is clearly illustrated in Figures 5.65, 5.66 and 5.67.

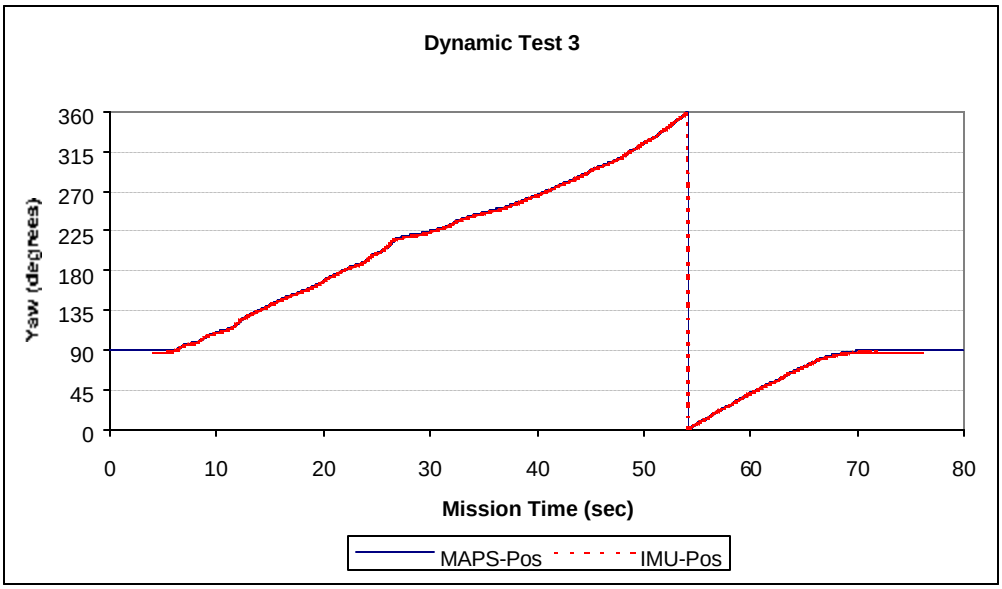


Figure 5.65: Dynamic Test 3 – Yaw vs. Time

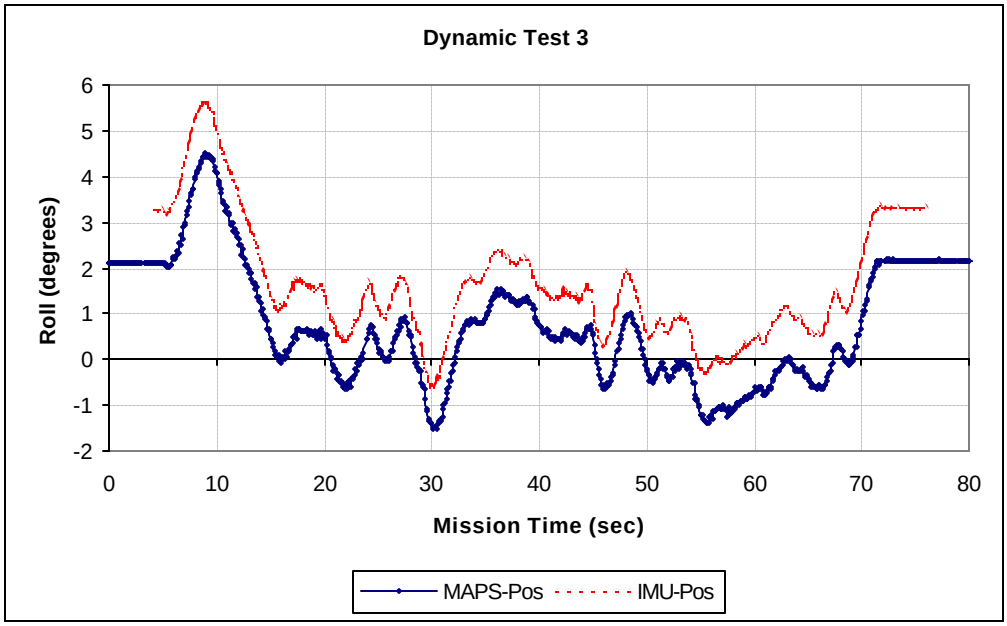


Figure 5.66: Dynamic Test 3 – Roll vs. Time

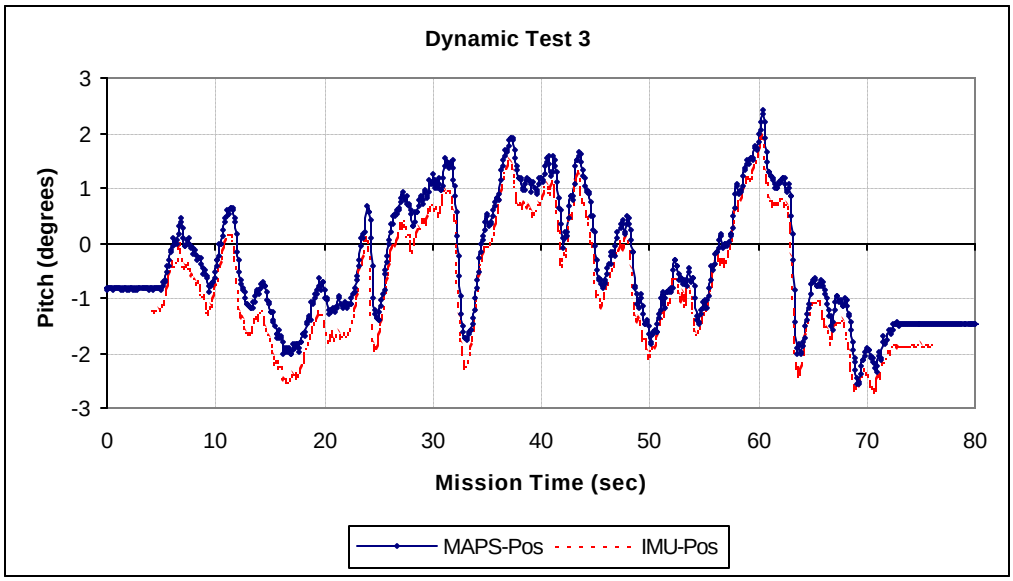


Figure 5.67: Dynamic Test 3 – Pitch vs. Time

The next three figures (5.68, 5.69 and 5.70) give the north, east and down velocity respectively. There are clearly drifts in the north and upward directions of the IMU, which contribute largely to the resulting position errors.

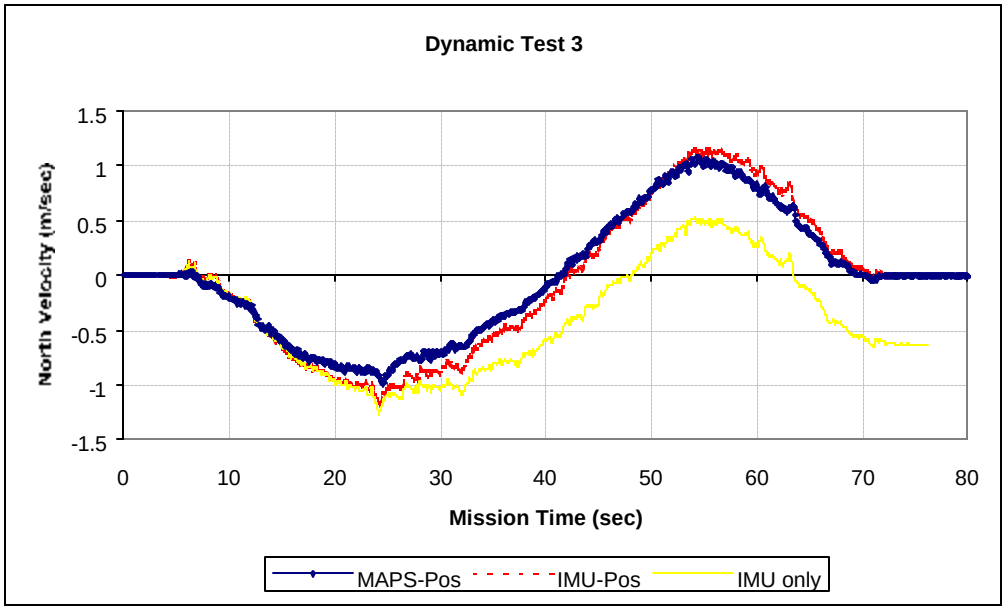


Figure 5.68: Dynamic Test 3 – North Velocity vs. Time

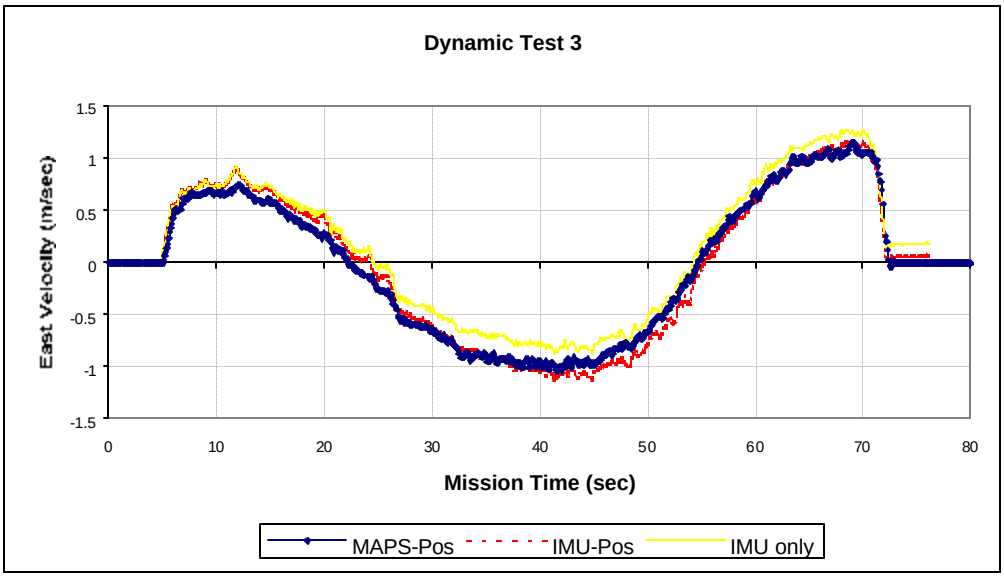


Figure 5.69: Dynamic Test 3 – East Velocity vs. Time

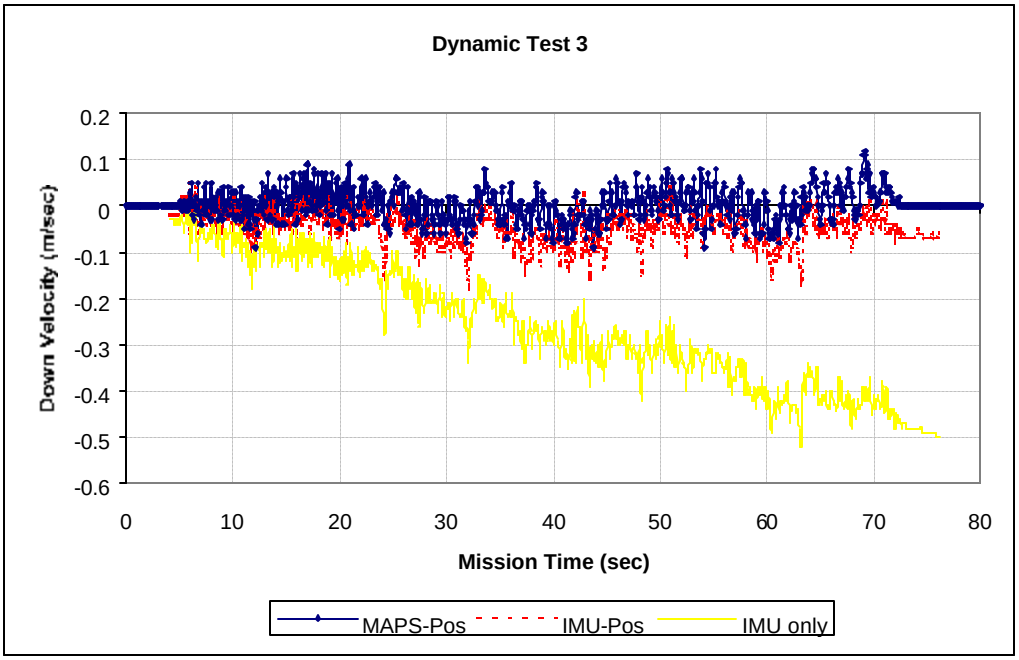


Figure 5.70: Dynamic Test 3 – Down Velocity vs. Time

Figures 5.71, 5.72 and 5.73 give the position, velocity and orientation errors respectively. The position error curves are a direct result of the velocity error curves. In the orientation error curve, the spikes in the yaw error may be attributed to slight timing

differences which cause the wrong yaw values to be matched at a certain time. It should be noted that the position and velocity errors are in Local Geodetic Vertical (LGV) coordinates defined by orthogonal axes of north, west and up.

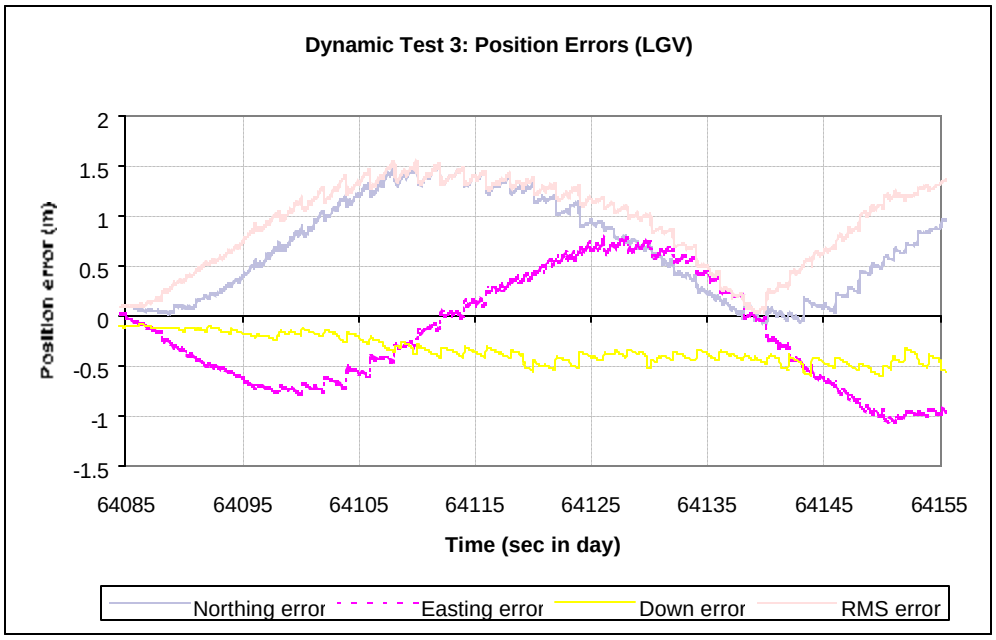


Figure 5.71: Dynamic Test 3 – Position Errors (LGV)

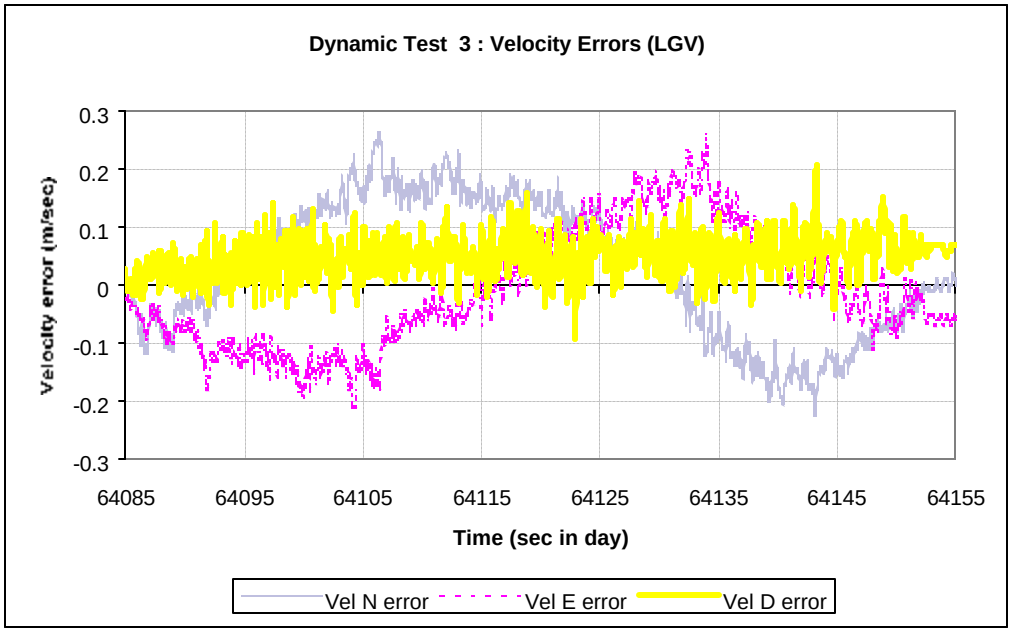


Figure 5.72: Dynamic Test 3 – Velocity Errors (LGV)

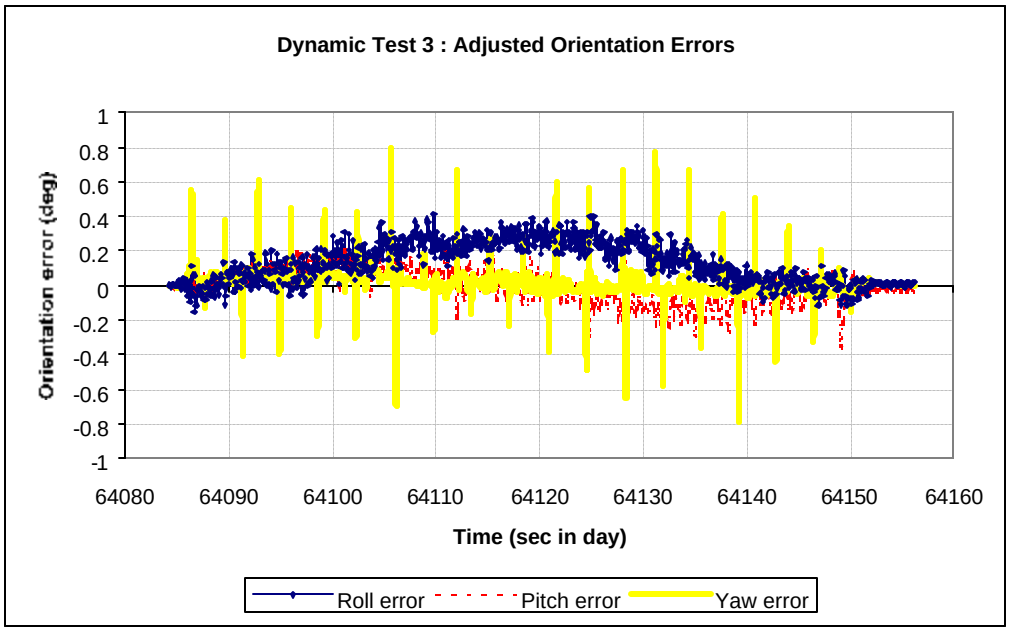


Figure 5.73: Dynamic Test 3 – Orientation Errors

Zigzag

The second combined dynamic test involves traversing in a zigzag pattern for about 80 seconds. This specific run definitively illustrates the effect of IMU position drift on the degradation of the KF performance. In Figure 5.74, the IMU/GPS position immediately diverges from the MAPS/GPS as the IMU navigation output has a huge eastwardly drift. The IMU/GPS KF is able to track the GPS position even with worsening IMU output but only to within 2-3 meters of the MAPS/GPS KF output.

Figures 5.75, 5.76 and 5.77 show the differences between the two systems in position, velocity and orientation respectively. In Figure 5.76, there is an immediate east velocity drift which causes IMU navigation to quickly diverge. Even as the velocities eventually settle down, the initial errors have already affected the position. The orientation errors remain relatively constant except for the yaw which drops nearly a degree from the start.

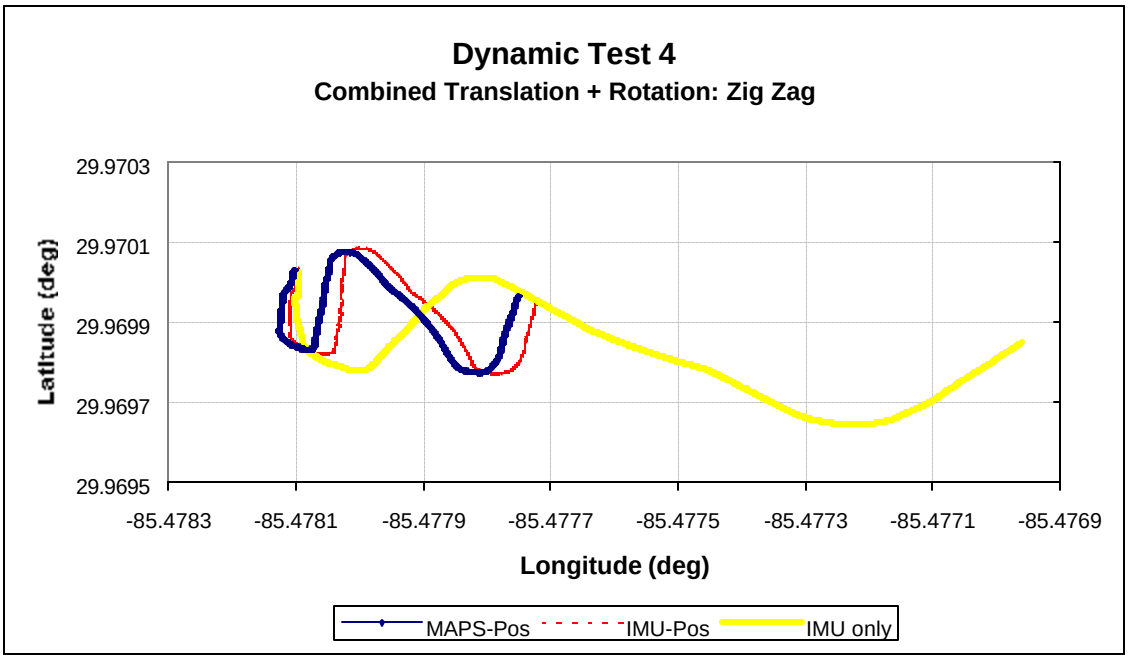


Figure 5.74: Dynamic Test 4 – Combined Translation and Rotation : Zigzag

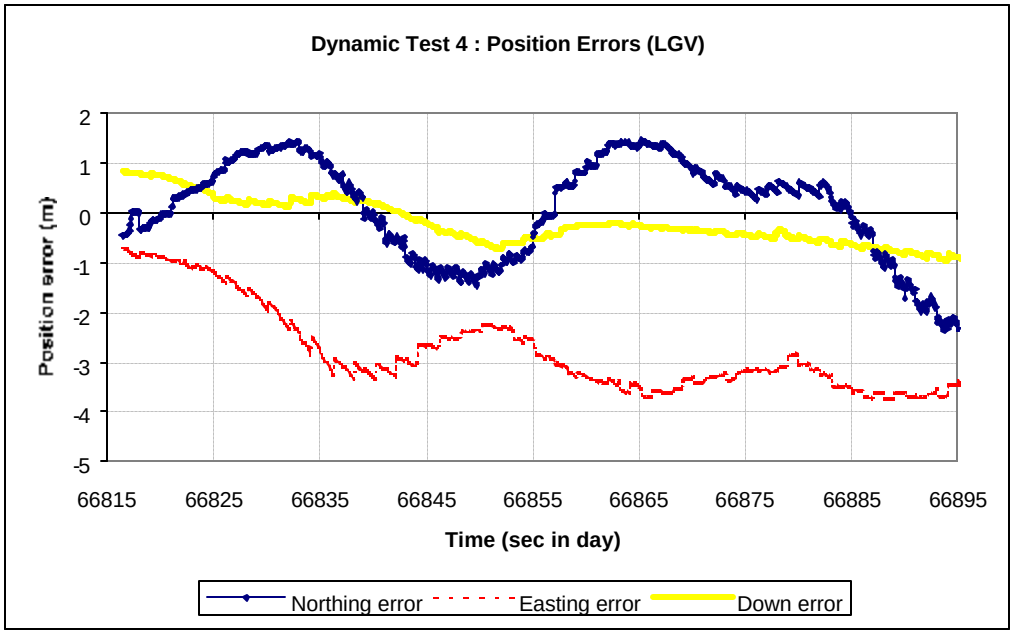


Figure 5.75: Dynamic Test 4 – Position Errors (LGV)

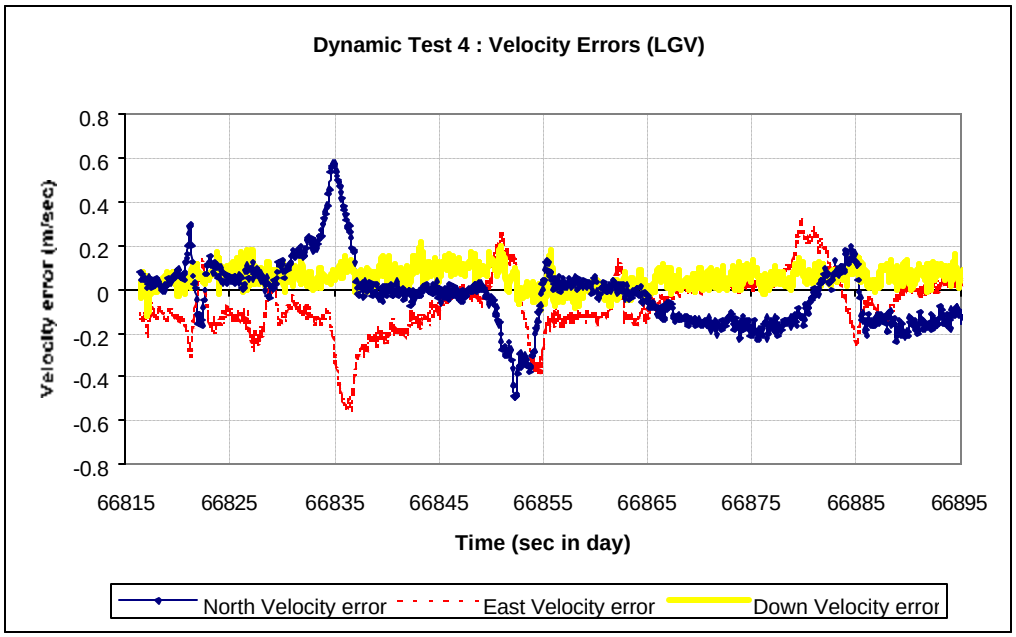


Figure 5.76: Dynamic Test 4 – Velocity Errors (LGV)

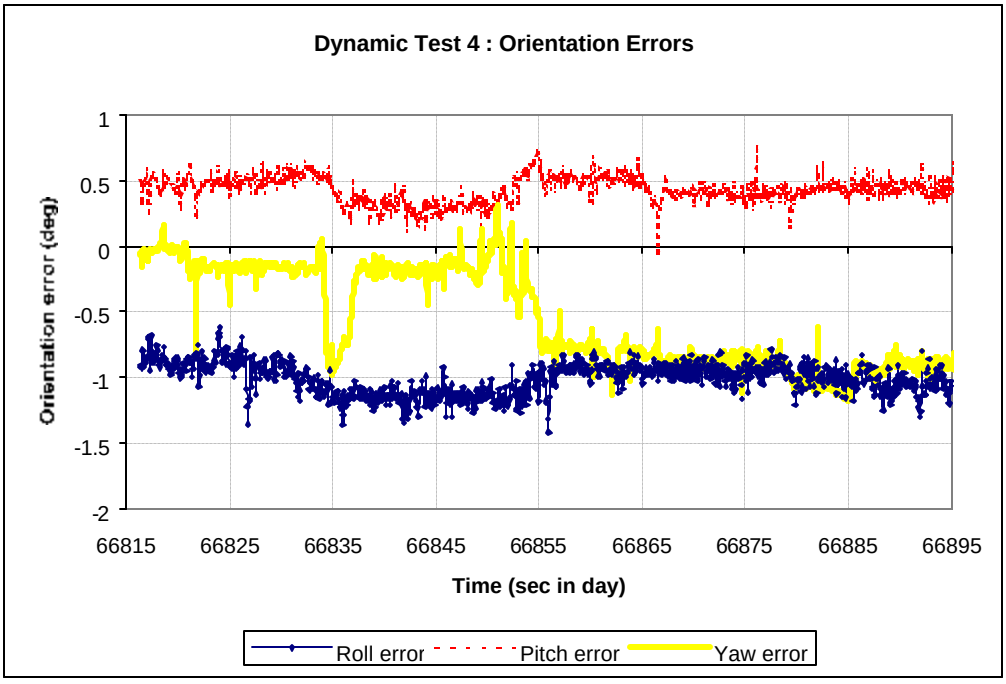


Figure 5.77: Dynamic Test 4 – Orientation Errors

Finally, Table 5.27 gives a summary of the errors of all four dynamic tests. These errors are defined as the difference between the MAPS/GPS KF values and the IMU/GPS KF values. It is clear that in pure translation, the standard deviation values for position, velocity and orientation are low. Also, the errors are lower in the direction of translation (east) mainly due to the accelerometer bias values being significantly lower in magnitude in the direction of motion. In pure rotation, the north and east velocities have high standard deviation which may be explained by the translation errors caused by the difference of location of the MAPS and the IMU relative to the vehicle control point. These patterns become apparent during the combined dynamic tests. Also, traveling at constant speed or turning at constant rotation results in lower error values.

Table 5.27: Summary of Dynamic Test Errors

Errors (MAPS/GPS - IMU/GPS)										
	Pos Rms	Northing	Easting	Altitude	Roll	Pitch	Yaw	Vel N	Vel E	Vel D
Dynamic Test 2 : Pure Translation										
AVE	0.44957	0.31241	-0.26993	-0.7497	-1.1549	0.3725	8.7584	0.02219	-0.0002	0.0607
STD	0.21681	0.26377	0.09543	0.29505	0.0415	0.0473	0.0424	0.03724	0.0255	0.0268
Dynamic Test 2 : Pure Rotation										
AVE	0.53053	0.17477	-0.39573	-0.3174	-1.0215	0.4248	1.1456	0.16057	-0.2507	0.0272
STD	0.43161	0.20375	0.49036	0.03912	0.221	0.2393	0.1987	0.48799	0.5577	0.0238
Dynamic Test 3: Combined Translation and Rotation (CW Loop)										
AVE	0.91134	0.67093	-0.18222	-0.3357	-1.0333	0.4136	2.2528	0.02955	-0.008	0.0503
STD	0.43206	0.48533	0.54655	0.13624	0.119	0.0982	0.1111	0.11404	0.102	0.037
Dynamic Test 3: Combined Translation and Rotation (Zigzag)										
AVE	2.95557	0.07274	-2.78014	-0.2124	-1.0122	0.4316	-0.556	-0.0281	-0.0691	0.0549
STD	0.8772	1.01874	0.8565	0.45831	0.1202	0.1016	0.3779	0.14706	0.1353	0.0492

CHAPTER 6
FUTURE WORK

The task of selecting and benchmarking lower-cost positioning systems as alternatives to the current MAPS/Ashtech positioning system on the Navigation Test Vehicle (NTV) has proven to be successful. The level to which both the Novatel Beeline DGPS and IMU/GPS positioning systems can be applied for autonomous vehicle navigation is highly dependent on the type of application. There are still some issues that will have to be looked into to increase the performance of the new systems.

Optimization of Beeline System Configuration

In the discussions in Chapter 4, it is stated that the Novatel Beeline DGPS system performed up to specifications. Once real-time kinematic position has converged, position accuracies fall within 10-20 cm CEP. However, the long-term operation of the Beeline is compromised by system resets attributed to processor overload. This is a direct result of the fast data log rates out of the remote receiver. In the tests conducted, position, velocity, and orientation data logs in ASCII are outputted at 5 Hz, which places a computational burden on the receiver processor. The solution is to reduce the data log rates down to 2 Hz. Also, if necessary, requesting logs in binary alleviates data traffic.

Future work may center on testing the effects of number of satellites in view, weather, time of day and even time of year. Another important thing to study is the effects of multipath.

The complications of using 2 separate GPS systems may have underlying effects. Lastly, latency tests can be performed to check for system time errors.

IMU/GPS Positioning System

At present the integration of the IMU with GPS through an external Kalman Filter has been completed. However, the system still needs some areas of improvement before it can fully be implemented on an autonomous ground vehicle.

Solidifying IMU Data Serial Interface

Much of the present system output problems stem from occasional errors in the IMU data. Error checking and data filtering are instant remedies but not solutions to the problem. To ensure accurate navigation output, the IMU data errors have to be eliminated completely.

The current system uses a DMA DOS driver for the ACB104 serial communications interface card. This driver is not guaranteed by the serial card manufacturer. Based on tests conducted, the serial card apparently experiences data overflows which results in system hang-up. Once the PC104 stack together with the ACB104 card is reset, the data link is re-established. Unpredictable behavior of the serial card and DOS DMA driver also causes bad data output. Since the IMU data is averaged from 100 Hz to 12.5 Hz, the actual occurrence of the erroneous data is difficult to pinpoint. Also, presently, the DOS DMA set-up so far is the only way that the IMU output has successfully been read.

To correct this problem, either the DOS DMA driver will have to be optimized, or the more reliable Windows DMA driver will have to be investigated. No DMA drivers for Linux are currently available.

A simpler solution would be to upgrade the software of the HG1700AG11 IMU so that it can output Inertial Messages (100 Hz) through an RS-422 asynchronous serial line at 115 Kbps. This entails sending the IMU back to Honeywell for a software upgrade (to a HG1700AG25 IMU) and subsequent recalibration. This eliminates the need for a separate serial interface card that can read in synchronous SDLC type messages at 1 Mbps. Most PC based platforms including the EBC-TX Plus are capable of directly reading in RS-422 serial data at 115 Kbps.

Implementing Fine Alignment

The fine alignment algorithm presented in Chapter 5 has been tested in simulation using static IMU data collected with reference to the MAPS orientation. The next step is to implement the same algorithm in C code and insert it in the NTV code as the succeeding process after the coarse alignment. It should be highly noted that the fine alignment is a crucial step in obtaining an accurate initial orientation from which the IMU navigation begins. However, the large yaw deviation seen in the coarse alignment tests makes the fine alignment ineffective. Again, this may be caused by the errors in the IMU data reception side. Further testing in comparison to the MAPS will have to be done in order to ensure that the IMU can perform self-alignment without any other external aid.

Optimization of Kalman Filter

The external Kalman filter (KF) has been tuned to the IMU given static and dynamic conditions. The Kalman filter uses nine states (three position error states, three velocity error states, and three orientation error states) making it ideal for modeling the MAPS. However, since the IMU accelerometers and gyroscopes are of a much lower grade, the biases and drifts

play a major role in calculating the position, velocity and orientation. To effectively model the IMU, a 15-state Kalman filter should be designed. This includes the nine states previously described plus three bias error states and three drift error states. Also, for better navigation solution integration, the IMU data output (accelerations and angular velocities) are fed directly into the KF, which passes its output into a navigation solution.

Redesigning Hardware Configuration

The first version of the IMU/GPS positioning system still leaves much to be redesigned. Among the higher priorities is selecting a new serial interface for the IMU. At present, the Sealevel ACB-104 is tying down the system since it uses an entire PC-104 stack just to read in IMU data at 100 Hz and output averaged data at 12.5 Hz. This is not only costly but inefficient. The switch over to Linux may present several options. The most immediate alternative would be to use the Sealevel ACB-104 and install a Linux Direct Memory Access (DMA) driver. Also, the PC-104 can be replaced by a SBC which drops system cost by nearly $2,000. In the near future, when SBC's improve in speed and memory efficiency, the two processors (IMU DP & PMP) may be merged into one. An optimized version of the IMU/GPS system may cost drop from $40,000 to $30,000 and will occupy only a half-shelf.

Using Novatel RT-2 GPS Receiver

The IMU/GPS positioning system shows typical position accuracy of 8-20 cm CEP when running under ideal conditions. Position converges continuously for up to 60 minutes and reaches accuracies as high as 4-5 cm. The slow convergence of the RT-20 becomes a major disadvantage when GPS lock is lost. Tests show that it takes normally 5 minutes for position to reconverge to a 0.5 m accuracy level. Tests have also shown that once GPS is lost, the KF is

able to maintain position to within 3 meters for not more than 30 seconds. Once GPS is recovered, position is used if the system position RMS is better than 1 meter.

The accuracy can be improved dramatically by replacing the Novatel RT-20 GPS receiver with a Novatel RT-2 GPS receiver. Both use the same technology of carrier-phase differential to provide high accuracy real-time kinematic position. However, the RT-2 achieves it extra accuracy and precision due to its being able to utilize dual-frequency measurements (L1/L2). This allows the RT-2 position to converge to within 2 cm. after 2 minutes (with an antenna baseline of 0.1 km) in kinematic mode. This becomes especially advantageous when the system is in a state of GPS recovery.

Another advantage of using an RT-2 as a replacement is the seamless transition. The RT-2 uses the same data logs, commands and functions (e.g. I/O pulse) the RT-20 uses. Thus, minimal change would be done to the GPS code and turn-over time is fast.

A big disadvantage, though, is the RT-2's high cost. Using RT-2 receivers for both base and remote stations will increase system cost by $8,000, making it self-defeating.

Evaluating Other Components

Aside from the Novatel RT-2, there are a lot of newer, lower-cost, high-performance GPS receivers to hit the market. A prime example of this is the Ashtech Z-Eurocard which sells for $10,000 and gives real-time kinematic accuracy of 1 cm at 1 Hz and 2 cm at 10 Hz. Receiving GPS updates at 10 Hz and outputting KF position at 20 Hz will allow the vehicle to navigate effectively at higher speeds (up to 20 m/s or 45 mph). The Ashtech Z-Eurocard is a bare GPS card which can be mounted and encased together with the other CPU boards, thus saving space and cost.

Novatel Black Diamond System

Novatel has recently come out with an integrated INS/GPS positioning system called the Black Diamond System (BDS, see Figure 6.1) that uses similar components in the Honeywell HG1700AG11 IMU and the Novatel RT-2 DGPS. The main differences with the IMU/GPS being developed are the level of integration and the GPS component used.

Figure 6.1: Novatel Black Diamond System.

First, Novatel is integrating at the lowest level, taking raw data from the IMU and the GPS receiver and fusing it by means of a 15-state Kalman Filter. By doing this, the BDS is capable of outputting position, velocity and attitude (PVA) at the IMU maximum output rate of 100 Hz. Also, the tightly-coupled integration minimizes system tuning due to the presence of one KF. The KF also uses the IMU sensor biases and drifts as states which greatly improves PVA calculation. Second, the BDS uses the Novatel RT-2 DGPS which gives 2 centimeter position accuracy as compared to the RT-20 which gives an average of 20 cm.

Other advantages of the BDS include packaging and sensor enclosures that require minimal installation. The sensor unit (top of Figure 6.1) and the controller (bottom of Figure 6.1) are separate, allowing the BDS sensor (IMU plus GPS antenna) to be located outside the vehicle while the BDS controller can be housed inside the vehicle. Also, the BDS uses the newer 600 series L1/L2 GPS antennas that weigh only a pound and already have a built-in choke ring.

A summary of the BDS performance specifications are given in Table 6.1.

Table 6.1: Novatel BDS Performance Specifications

Specification	Performance
Position Accuracy	Stand Alone <3 m RMS Code Differential 0.25 to 1 m RMS RT-20 0.05 to 1 m RMS RT-2 0.02 m RMS Post Processed 0.02 to 2 m RMS
Velocity Accuracy RMS	0.02 m/s (nominal)
Attitude Accuracy RMS	Roll 0.015° Pitch 0.015° Azimuth 0.05°
Acceleration Accuracy RMS	0.03 m/s
Time Accuracy	50 ns
Time to First Fix	120 s
Reacquisition	2 s
Position Degradation w/ Loss of GPS	
After 1 Minute	< 4 meters
After 2 Minutes	< 17 meters
After 3 Minutes	< 38 meters
Alignment Time	
RT-2 GPS	2-3 minutes
IMU	5 minutes
Data Rates	Raw Measurement 100 Hz Computed Position 100 Hz Computed Attitude 100 Hz
Measurement Precision	L1 C/A Code 6 cm RMS L1 Differential Phase 0.75 mm RMS L2 P Code 23 cm RMS L2 Differential Phase 2 mm RMS
Dynamics	Acceleration < 50 g Velocity < 1000 knots Height < 60,000 ft Rotation < 1000 deg/s
Channels	L1 - 12 L2 - 12

APPENDIX A
SUPPLEMENTAL MATHEMATICAL EQUATIONS

Fourth Order Runge Kutta

Position

$$X' = X + \tfrac{1}{6}(k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x}) \tag{5.83}$$

where,

$k_{1x} = f(X)$, using Eq. (5.81)

$k_{2x} = f(X + \tfrac{1}{2}k_{1x})$, using Eq. (5.81)

$k_{3x} = f(X + \tfrac{1}{2}k_{2x})$, using Eq. (5.81)

$k_{4x} = f(X + k_{3x})$, using Eq. (5.81)

Orientation

$$q' = q + \tfrac{1}{6}(k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x}) \tag{5.93}$$

where,

$k_{1x} = f(q)$, using Eq. (5.92)

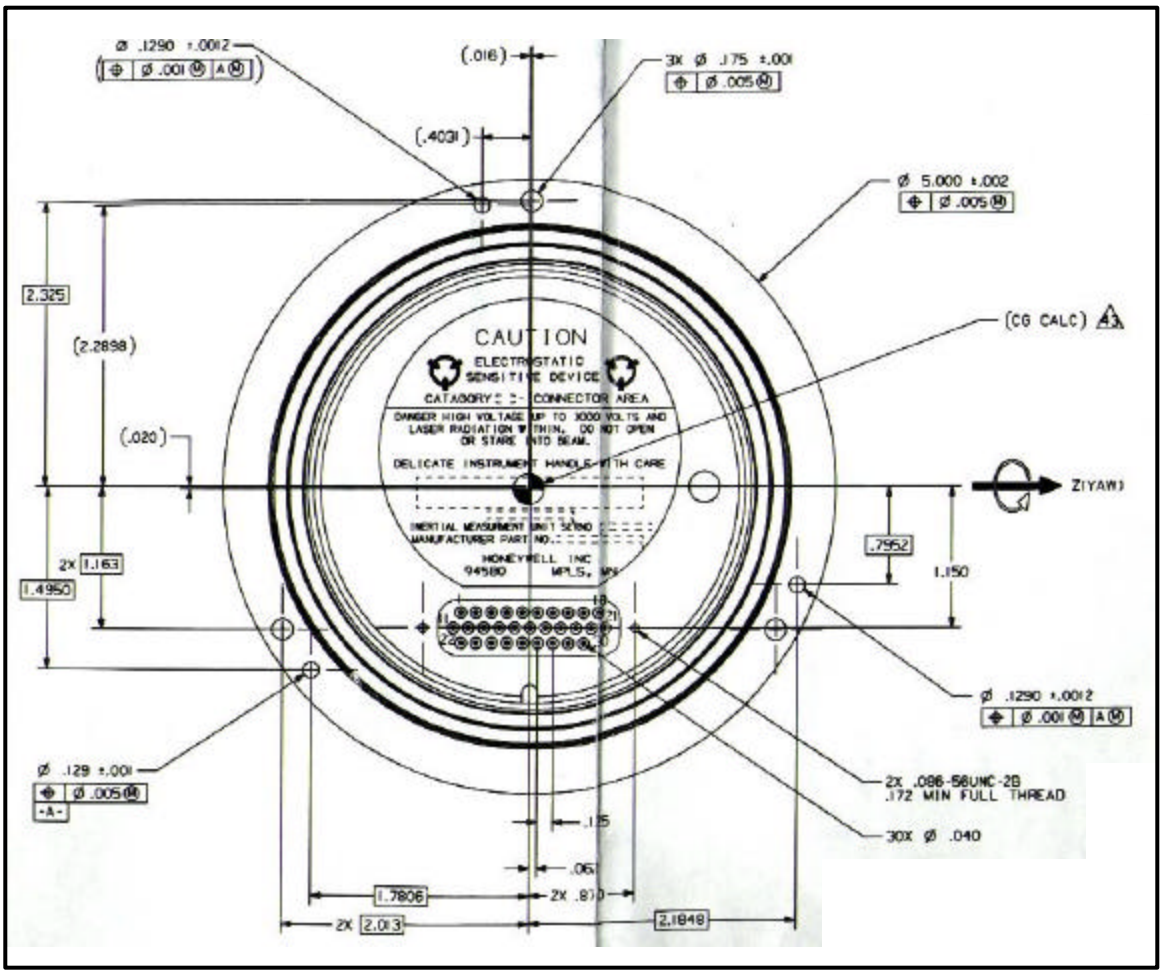
$k_{2x} = f(q + \tfrac{1}{2}k_{1x})$, using Eq. (5.92)

$k_{3x} = f(q + \tfrac{1}{2}k_{2x})$, using Eq. (5.92)

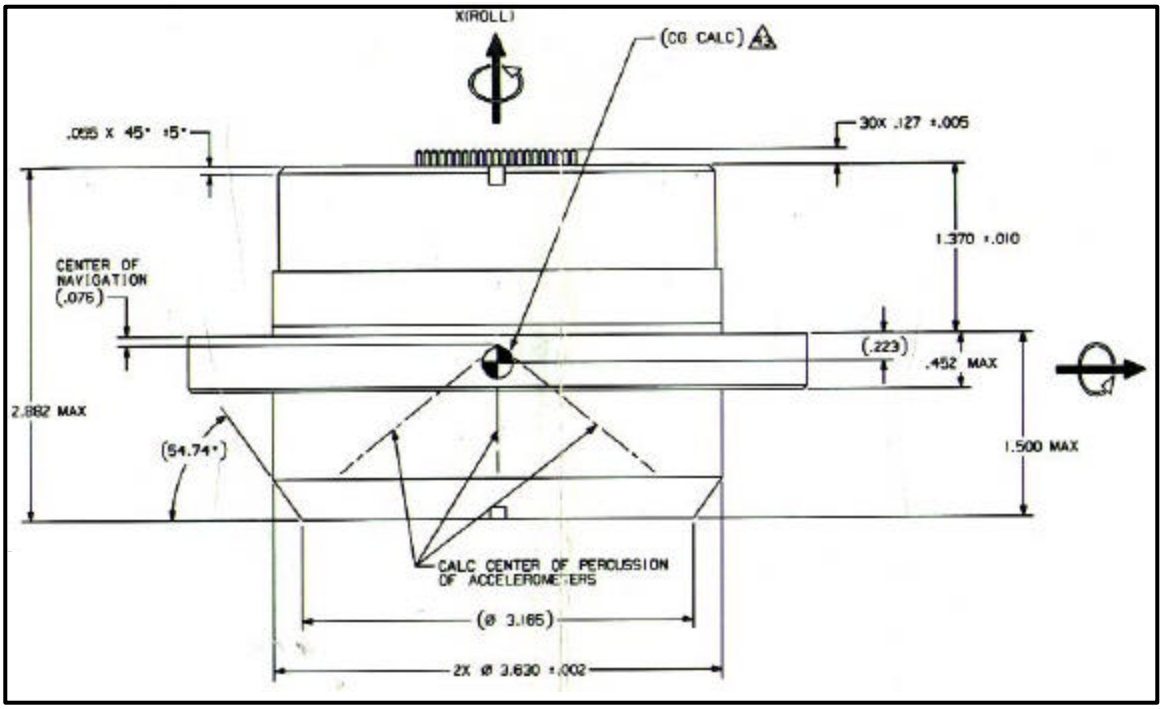
$k_{4x} = f(q + k_{3x})$, using Eq. (5.92)

APPENDIX B
HONEYWELL HG1700AG11 SCHEMATIC DRAWINGS

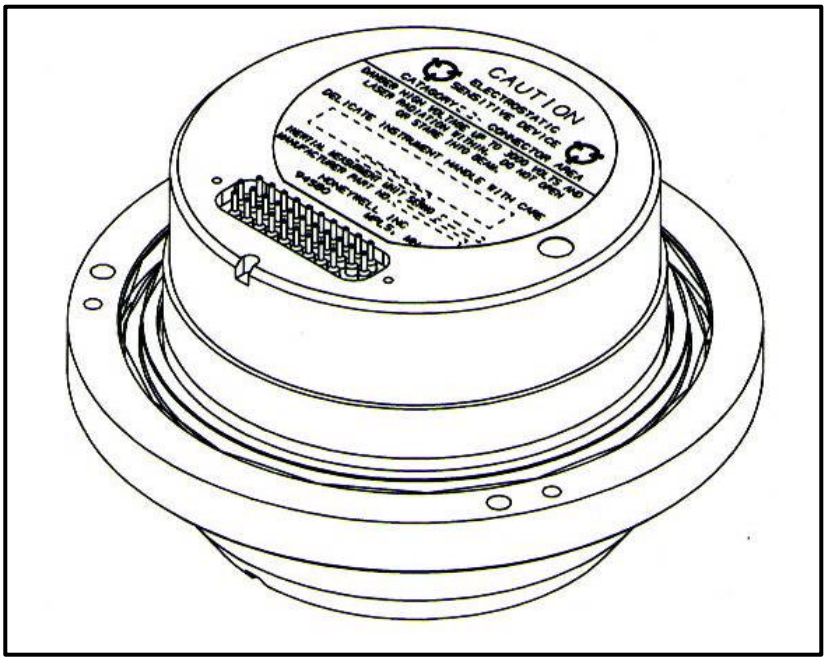
Front View



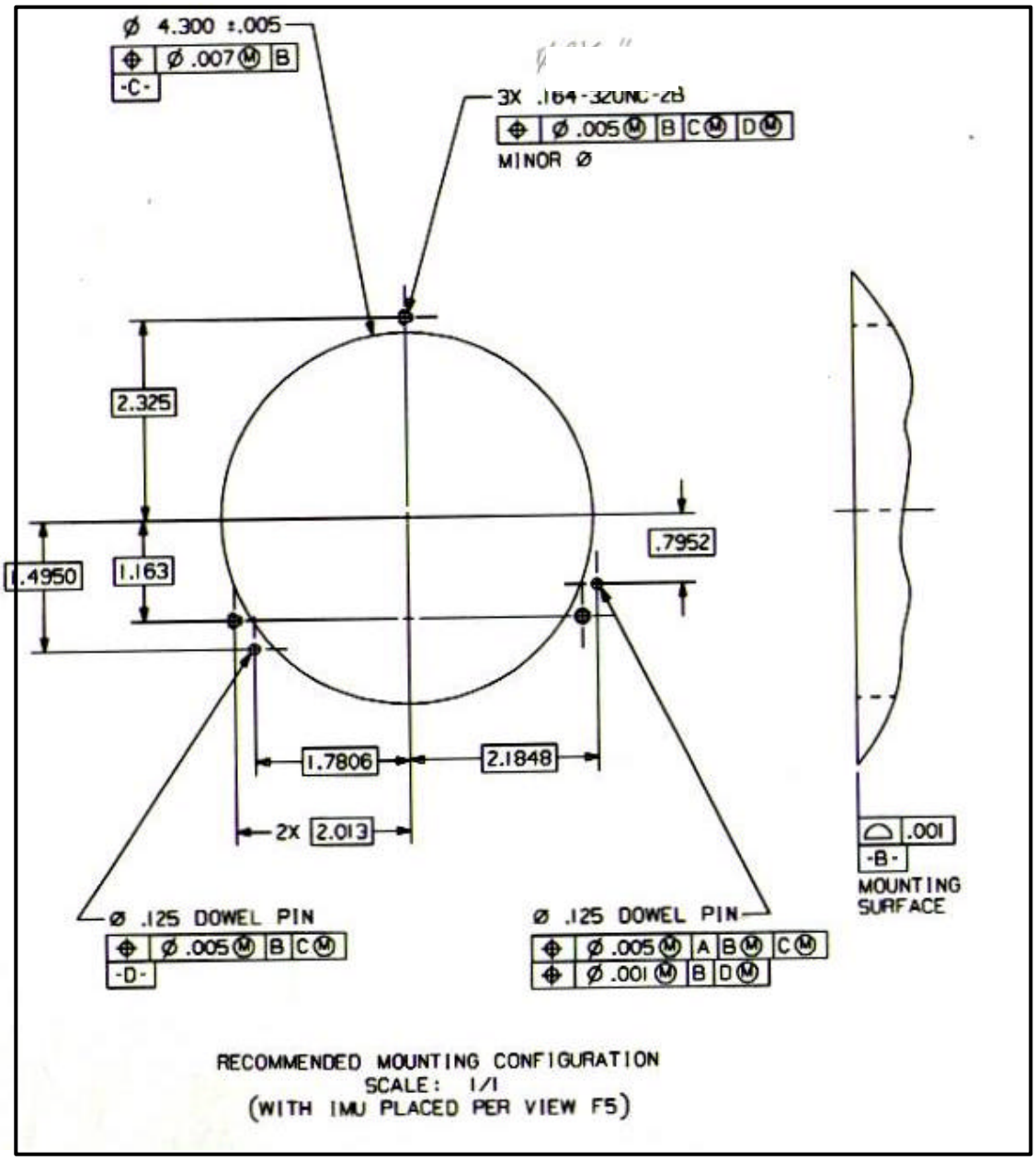
Top View



Isometric View



Mounting Holes Layout



APPENDIX C
COMPUTER CODE

```
/*-------------------------------------------------------------------------------------*/
/* library     : pos.c                                                                 */
/* Version     : 1.0                                                                   */
/* Contractor  : Center for Inteligent Machines and Robotics                           */
/*             : University of Florida                                                 */
/* Project     : Autonomous Vehicle System                                             */
/*-------------------------------------------------------------------------------------*/
/* Date        : 05/01/01 original creation                                            */
/* Last Update : 06/24/01 switch from MAPS to IMU                                      */
/*-------------------------------------------------------------------------------------*/
/* This program spawns off threads to read in IMU and GPS data and sends               */
/* it to the Kalman filter.  Sensor offsets are also read in                           */
/* This pos code is JAUGS compatible and uses the latest MRS                           */
/*-------------------------------------------------------------------------------------*/

#include <pthread.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <unistd.h>

#define MAIN_PROGRAM
#include "pos.h"
#include "imuLib.h"
#include "filterLib.h"
#include "gpsLib.h"
#include "posConsole.h"
#include "portOpsLib.h"
#include "mrtLib.h"
#include "JAUGSMsgLib.h"
#include "txRxCoreJAUGSMsgLib.h"
#include "txRxPositionMsgLib.h"

#define USE_PATCH 0
#define FALSE 0
#define TRUE  1
#define MAX_GPS_RMS 0.5
#define DEBUG_STATE 99
#define SEC_IN_DAY 86400
```

```
pthread_t consoleThreadId;
pthread_t posThreadId, readGpsThreadId, updateStateThreadId;
unsigned char myNodeId, mySubsystemId;
unsigned char desiredState, currentState;
position_t novatel, imu, kFilter, ref;
imuInfo_t imuInfo;
novatelInfo_t novatelInfo;
double hertz=0;
double missionTime=0;
int setClockCount=0;
double dt1=0.0, dt2=0.0;
double gps_to_inu[3], inu_to_cp[3], gps_to_cp[3];
FILE *KFfp,*IMUfp,*GPSfp,*ERRfp;
int filesave = 0;
int USE;
double refRoll,refPitch,refYaw;
double biasX,biasY,biasZ,driftX,driftY,driftZ;
int TIME_COARSE;
double driftNorth,driftEast,driftDown,driftRms=0.0;
double imuNorth,imuEast,imuDown,imuRms=0.0;
double driftRoll,driftPitch,driftYaw=0.0;
double startroll,startpitch,startyaw=0.0;
int STATIC_TEST = TRUE;
double time1start,time1stop,time2start,time2stop,time3start,time3stop;
int WITHHOLD_GPS;
int HARDCODE_GPS;
double HClat,HClon,HCalt;
int NAV_UPDATE;

double geoDist(position_t pos0, position_t pos1)
{
        double dlat, dlon;
        double dn, de;
        double dist;
        dlat = pos1.lat - pos0.lat;
        dlon = pos1.lon - pos0.lon;
        dn = dlat*D2R * earth_radius;
        de = dlon*D2R * (earth_radius * cos(pos0.lat*D2R));
        dist = sqrt(dn*dn + de*de);
        return dist;
}

void *updateStateThread(void *threadData)
{
        int mode, updateCount;
        double dist;

        while (1)
        {
                pthread_testcancel();
```

```
		if (currentState != desiredState)
		{
			if (desiredState == READY_STATE)
			{
				currentState = INITIALIZE_STATE;
				/* wait for new gps data */
				novatelInfo.newDataFlag = 0;

			while(novatelInfo.newDataFlag == 0)
				{
					pthread_testcancel();
					sleep(1);
				}
				/* wait for good gps data - assumes good gps for rest of
initialization */
				while(novatel.positionRms > MAX_GPS_RMS)
				{
					pthread_testcancel();
				sleep(1);
				}
			/* wait for good IMU data */
				while((mode =
getImuData(&imu,&imuInfo,kFilter,novatel,novatelInfo)) == -1)
					pthread_testcancel();
				/* check if IMU is trying to align */
				while (mode == 1) /* IMU is aligning */
				{
					pthread_testcancel();
					/* wait for good IMU data */
					while((mode =
getImuData(&imu,&imuInfo,kFilter,novatel,novatelInfo)) == -1)
						pthread_testcancel();

					/* make sure IMU data is close to GPS data */
					updateCount = 0;
					 dist = geoDist(novatel, imu);
			while(dist > 0.5)
					{
						imu.lat = novatel.lat;
						imu.lon = novatel.lon;
						imu.alt = novatel.alt;
						sleep(2);
						pthread_testcancel();

						/* wait for good IMU data */
						while((mode = getImuData(&imu,
&imuInfo,kFilter,novatel,novatelInfo)) == -1)
							pthread_testcancel();

						dist = geoDist(novatel, imu);
				dist = 0.1;
```

```
                                        updateCount++;

                                        if ((dist > MAX_GPS_RMS) && (updateCount
> 10)) /* IMU not accepting update so restart */
                                        {
                                                imuInfo.status[0] = START_UP;
                                                imuInfo.status[1] = RESET;
                                  sleep(5);
                                                pthread_testcancel();
                                        }
                                }
                        }

                        /* make sure IMU data is close to GPS data */
                        updateCount = 0;
                        dist = geoDist(novatel, imu);

                        while(dist > 0.5)
                        {
                                imu.lat = novatel.lat;
                                imu.lon = novatel.lon;
                                imu.alt = novatel.alt;
                                sleep(2);
                                pthread_testcancel();

                                /* wait for good IMU data */
                                while((mode =
getImuData(&imu,&imuInfo,kFilter,novatel,novatelInfo)) == -1)
                                        pthread_testcancel();

                                dist = geoDist(novatel, imu);
                                updateCount++;

                                if ((dist > 0.5) && (updateCount > 10)) /* IMU not
accepting update so restart */
                                {
                                        imuInfo.status[0] = START_UP;
                                        break;
                                }
                        }

                        if (dist < 0.5)
                        {
                                /* put IMU in ready state */
                                currentState = READY_STATE;
                        }
                        else
                {
                        /* unable to initialize */
                                currentState = desiredState = FAILURE_STATE;
                        }
```

```
}
else if (desiredState == STANDBY_STATE)
{
	/* wait for position update request */
	updateCount = 0;
	while (!(imuInfo.status[0] == POS_UPDATE_REQUEST))
	{
		pthread_testcancel();
		updateCount++;
		sleep(5);

		if (updateCount >= 10)
			break;
	}

	if (updateCount < 10)
	{
		/* update IMU position */
		imu.lat = novatel.lat;
		imu.lon = novatel.lon;
		imu.alt = novatel.alt;
		currentState = STANDBY_STATE;
	}
	else
	{
		/* unable to standby */
		currentState = desiredState = FAILURE_STATE;
	}
}
else if (desiredState == SHUTDOWN_STATE)
{
	/* wait for position update request */
	updateCount = 0;

	while (!(imuInfo.status[0] == POS_UPDATE_REQUEST))
	{
	pthread_testcancel();
		updateCount++;
		sleep(5);
		if (updateCount > 10)
			break;
	}

	/* update IMU position */
	imu.lat = novatel.lat;
	imu.lon = novatel.lon;
	imu.alt = novatel.alt;
	sleep(2);

	/* shutdown IMU */
	/* always go into shutdown state - even if it did not work */
```

```
                                currentState = SHUTDOWN_STATE;
                        }
                }
                else
                        usleep(1e6);
        }
        return NULL;
}

#define SET_CLOCK_COUNT 120
#define BUF_SIZE 255

void *readGpsThread(void *threadData)
{
        struct timeval t0, t1, tNow;
        struct timezone z;
        port_t gpsDataPort;
        FILE *fp;
        char buf[BUF_SIZE], header[6];
        int week, cmStatus;
        double seconds, offset, offsetStd, utcOffset;
        double latency,age,horSpd,trkGnd,vertSpd;
        int solStatus, velStatus, rt20Status, stnId, numSats,datumId;
        double lat,lon,hgt,latStd,lonStd,hgtStd,undulation;
        int settingClock=0;
        double secondsBeforeToday, gpsSecondsToday;
        double gpsTime, gpsPosTime, gpsVelTime;
      int clock_sec_of_day;

        gpsDataPort = initPort(1,19200,"8N1");
        flushBuf(gpsDataPort,BOTH_BUF);
        fp=fdopen(gpsDataPort,"r");
        resetMarkPulse();

        /* set up gps reciever */
        while(1)
        {
                pthread_testcancel();

            if (HARDCODE_GPS)
         { /* HARD CODE POSITION */
                gettimeofday(&tNow,&z);
           clock_sec_of_day = tNow.tv_sec % 86400;
                novatel.time = (double)clock_sec_of_day + (double)tNow.tv_usec / 1.0e6 -
0.25*0;
        novatel.lat = HClat;
        novatel.lon = HClon;
        novatel.alt = HCalt;
        novatelInfo.newDataFlag = 1;
        novatel.positionRms = 0.1;
        sleep(1) ;
```

```
        }

        else
        {
                fgets(buf,BUF_SIZE,fp);
                pthread_testcancel();

          if (strncmp(buf,"$MKTA",5) == 0)
                {
                        sscanf(buf,"%5s,%d,%lf,%lf,%lf,%lf,%d",

      header,&week,&seconds,&offset,&offsetStd,&utcOffset,&cmStatus);

                        if (settingClock)
                        {
                                settingClock=0;
                                secondsBeforeToday = (t0.tv_sec /
(long)SEC_IN_DAY)*SEC_IN_DAY;
                                gpsTime = seconds - offset;
                                gpsSecondsToday = (double)((int)gpsTime %
(int)SEC_IN_DAY)
                                        + (double)(gpsTime - (int)gpsTime);
                                gettimeofday(&t1, &z);
                                gpsSecondsToday += (double)(t1.tv_sec-t0.tv_sec)
                                        + (double)(t1.tv_usec-t0.tv_usec)/1.0e6;

                                tNow.tv_sec = secondsBeforeToday + (long)gpsSecondsToday;
                                tNow.tv_usec = (long)((gpsSecondsToday -
(long)gpsSecondsToday)*1e6);
                                settimeofday(&tNow,&z);
                        }
                }

                else if (strncmp(buf,"$VLHA",5) == 0)
                {
                        sscanf(buf,"%5s,%d,%lf,%lf,%lf,%lf,%lf,%lf,%d,%d",

      header,&week,&seconds,&latency,&age,&horSpd,&trkGnd,&vertSpd,&solStatus,
                                &velStatus);

                        if ((solStatus == 0) && (velStatus < 2))
                        {
                                gpsVelTime = (int)seconds % SEC_IN_DAY;
                                gpsVelTime += (seconds - (int)seconds);
                                gpsVelTime -= latency;
                                novatel.velN = horSpd * cos(trkGnd * D2R);
                                novatel.velE = horSpd * sin(trkGnd * D2R);
                                novatel.velD = -vertSpd;
                        }
                }
```

```
                else if (strncmp(buf,"$RT20A",6) == 0)
                {

        sscanf(buf,"%6s,%d,%lf,%d,%lf,%lf,%lf,%lf,%d,%lf,%lf,%lf,%d,%d,%d",

        header,&week,&seconds,&numSats,&lat,&lon,&hgt,&undulation,

        &datumId,&latStd,&lonStd,&hgtStd,&solStatus,&rt20Status,&stnId);

                        gpsPosTime = (int)seconds % SEC_IN_DAY;
                        gpsPosTime += (seconds - (int)seconds);
                        novatel.time = gpsPosTime;
                        novatel.lat = lat;
                        novatel.lon = lon;
                        novatel.alt = hgt;
                        novatel.positionRms = sqrt(latStd*latStd+lonStd*lonStd);

                if ((solStatus == 0) && (rt20Status < 2))
                                novatelInfo.newDataFlag = 1;
                }
                else
                        printf("%s\r\n\n",buf);

                setClockCount++;
                if(setClockCount >= SET_CLOCK_COUNT)
                {
                        setClockCount = 0;
                        settingClock=1;
                        gettimeofday(&t0,&z);
                        outputMarkPulse();
                }
        }
        }
}

void *posThread(void *threadData)
{
        struct timeval t0, t1;
        struct timezone z;
      static double startTime;
        double timePrev=0.0;
        float yawPrev=0.0;
        int new_gps_data, new_inu_data = 1;
        int initialized=FALSE;
        int ready;
        int pos_update, skip_update;
        INU_DATA inu, inu_past, delta_inu, inu_dot;
        QUEUE q;
        GPS_DATA gps, gps_past, delta_gps;
        NC_DATA nc;
        double Phi[number_states][number_states];
```

```
    double P[number_states][number_states];
    double x[number_states];
    double Q[number_states];
    double H[number_meas][number_states];
    double R[number_meas][number_meas];
    double F[number_states][number_states];
    double PHT[number_states][number_meas];
    double Resid_Cov[number_meas][number_meas];
    double meas_estimate[number_meas];
    double Gain[number_states][number_meas];
    double Residual[number_meas];
    double specific_force[3];
int i,j;
double x_ic[number_meas];
double P_ic[number_meas][number_meas];
imuInfo.errorCount = 0;

    while(1)
    {
            pthread_testcancel();
            if ((desiredState == READY_STATE) && (currentState == READY_STATE))
            {
                    gettimeofday(&t0,&z);
                    /* get new imu data */
                    while (getImuData(&imu, &imuInfo, kFilter, novatel, novatelInfo) != 0)
                            pthread_testcancel();
                    /* check data */
                    if ( ((int)(ref.lat - imu.lat) == 0) &&
                                    ((int)(ref.lon - imu.lon) == 0) )
           {
                    /* kalman Filter */
                            inu.time      = imu.time ;
                            inu.latitude  = imu.lat;
                            inu.longitude = imu.lon;
                            inu.altitude  = (double)imu.alt;
                            inu.roll      = (double)imu.roll;
                            inu.pitch     = (double)imu.pitch;
                            inu.yaw       = (double)imu.yaw;
                            inu.vel_north = (double)imu.velN;
                            inu.vel_east  = (double)imu.velE;
                            inu.vel_down  = (double)imu.velD;

                            /* read new gps data */
                            new_gps_data = novatelInfo.newDataFlag;
                            if (new_gps_data)
                            {
                                    gps.time      = novatel.time;
                                    gps.latitude  = novatel.lat;
                                    gps.longitude = novatel.lon;
                                    gps.altitude  = (double)novatel.alt;
                                    /* decide if to use gps data */
```

```
                        if (novatel.positionRms > MAX_GPS_RMS)
                    {   new_gps_data = FALSE ;
                 printf("BAD GPS! ");
               }
               /* WITHHOLD GPS ROUTINE */
               if (WITHHOLD_GPS)
               {
                 if (missionTime>time1start && missionTime<time1stop)
                 {   new_gps_data = FALSE ;
                    printf("LOST GPS! ");
                 }
                 else if (missionTime>time2start && missionTime<time2stop)
                 {   new_gps_data = FALSE ;
                    printf("LOST GPS! ");
                 }
                 else if (missionTime>time3start && missionTime<time3stop)
                 {   new_gps_data = FALSE ;
                    printf("LOST GPS! ");
                 }
                 else
                    printf("          ");
               }
                        novatelInfo.newDataFlag = 0;
                    }
                    if (!initialized)
                    {
               startTime = inu.time;
                        if (new_gps_data)
                        {

INU_initialize(&pos_update,inu,gps,&nc,&q,&inu_past,
                                                    &gps_past,Phi,F,P,Q,x,H,R);
                    initialized = TRUE;
                 startroll = imu.roll;
                 startpitch = imu.pitch;
                 startyaw = imu.yaw;
                        }
                    }
                    else
                    {
                    ready = INU_setup_data(&inu,new_inu_data,gps,new_gps_data,

&q,&inu_past,&delta_inu,&gps_past,&delta_gps);
                        if (ready == TRUE)
                        {
                                if ((new_gps_data)||(NAV_UPDATE))
                                {
                    /*-------------------------------------------------------------*/
                    /*                    Kalman Filter                    */
                    /*-------------------------------------------------------------*/
                    if (NAV_UPDATE)
```

```
                                    {
                                      printf("NAV UPDATE\n!");
                                      delta_inu.latitude = 0;
                                      delta_inu.longitude = 0;
                                      delta_inu.altitude = 0;
                                      delta_inu.vel_north = 0;
                                      delta_inu.vel_east = 0;
                                      delta_inu.vel_down = 0;

                                      INU_initialize(&pos_update,inu,gps,&nc,&q,&inu_past,
                                                                  &gps_past,Phi,F,P,Q,x,H,R);
                                      pos_update = 0;
                                    }

                                    form_INU_system_error_model (inu,delta_inu,&inu_dot,gps,
                                                        delta_gps,specific_force,F) ;
                                                        form_INU_state_transition
(F,delta_gps,Phi) ;

                                    propagate_INU_state (Phi,x) ;
                                    propagate_INU_covariance (inu,delta_inu,delta_gps,Phi,Q,P) ;
                                     INU_setup_measurements
(&pos_update,inu,delta_inu,gps,delta_gps,

        gps_to_inu,H,Residual,R) ;
                                     compute_INU_residual_covariance (P,H,R,PHT,Resid_Cov) ;
                                      compute_INU_meas_estimate (H,x,meas_estimate) ;
                                   skip_update = test_measurement_quality
(Residual,meas_estimate,
                                                                        Resid_Cov) ;

                                                        if (skip_update == FALSE)
                                                        {
                                                        update_INU_covariance
(Resid_Cov,PHT,Gain,H,P) ;
                                                        update_INU_state
(Gain,Residual,meas_estimate,x) ;
                                     }
                                     /*-------------------------------------------------------------*/
                                              /*        END OF KALMAN FILTER
*/
                                     /*-------------------------------------------------------------*/
                                    }
                                                  if (NAV_UPDATE)
                                  NAV_UPDATE = FALSE;
                                 else
                                 {
                                   compute_INU_nav_solution (new_gps_data,&q,delta_inu,inu,
                                                        inu_dot,specific_force,x,P,inu_to_cp,&nc) ;
        }
                                                        if (USE_PATCH)
```

```
patch(new_gps_data,gps_to_cp,gps,&nc) ;

                                        /* write data to shared memory */
                                                kFilter.time = nc.time;
                                                kFilter.lat  = nc.latitude;
                                                kFilter.lon  = nc.longitude;
                                                kFilter.alt  = (float)nc.altitude;
                                                if (q.ptr != 0)
                                                {
                                                        kFilter.roll  = (float)q.inu[q.ptr].roll;
                                                        kFilter.pitch = (float)q.inu[q.ptr].pitch;
                                                        kFilter.yaw   = (float)q.inu[q.ptr].yaw;
                                                }
                                                kFilter.velN = (float)nc.vel_north;
                                                kFilter.velE = (float)nc.vel_east;
                                                kFilter.velD = (float)nc.vel_down;
                                                kFilter.positionRms = (float)nc.rms;
                                                /* calculate yaw rate */

                                                if(timePrev != 0.0)
                                                        kFilter.yawRate = (kFilter.yaw -
yawPrev) /
                                                                        (kFilter.time -
timePrev);
                                                else
                                                        kFilter.yawRate = 0.0;
                                                timePrev = kFilter.time;
                                                yawPrev = kFilter.yaw;
                                        }
                                }
                        }
                        /* calculate loop rate */
                        gettimeofday(&t1,&z);
                        hertz = 1.0 / ((double)(t1.tv_sec - t0.tv_sec)
                                        + (double)(t1.tv_usec - t0.tv_usec)/1.0e6);
                missionTime = inu.time - startTime;

                /* Save Data to File */
                if ((filesave == TRUE) && (missionTime > 4.0))
                {   fprintf(KFfp,"%.2f %.7f %.7f %.2f %.2f %.2f %.2f %.2f %.2f %.2f\n
",missionTime,kFilter.lat,kFilter.lon,kFilter.alt,kFilter.roll*R2D,kFilter.pitch*R2D,kFilter.yaw*R
2D,kFilter.velN,kFilter.velE,kFilter.velD);

                  fprintf(IMUfp,"%.2f %.7f %.7f %.2f %.2f %.2f %.2f %.2f %.2f %.2f\n
",missionTime,imu.lat,imu.lon,imu.alt,imu.roll*R2D,imu.pitch*R2D,imu.yaw*R2D,imu.velN,im
u.velE,imu.velD);

                  fprintf(GPSfp,"%.2f %.7f %.7f %.2f %.2f %.2f %.2f\n
",missionTime,novatel.lat,novatel.lon,novatel.alt,novatel.velN,novatel.velE,novatel.velD);
```

```
                    fprintf(ERRfp,"%.2f %.3f %.3f %.3f %.3f %.2f %.2f %.2f %.3f %.3f %.3f
%.3f\n
",missionTime,driftNorth,driftEast,driftDown,driftRms,driftRoll*R2D,driftPitch*R2D,driftYaw*
R2D,imuNorth,imuEast,imuDown,imuRms);
                }

                if (STATIC_TEST)
                {
                    driftNorth = (novatel.lat-kFilter.lat)*D2R * earth_radius;
                    driftEast = (novatel.lon-kFilter.lon)*D2R * (earth_radius *
cos(novatel.lat*D2R));
                   driftDown = novatel.alt - kFilter.alt;
                   driftRms = sqrt(driftNorth*driftNorth+driftEast*driftEast);
                   driftRoll = startroll - kFilter.roll;
                   driftPitch = startpitch - kFilter.pitch;
                   driftYaw = startyaw - kFilter.yaw;
                    imuNorth = (novatel.lat-imu.lat)*D2R * earth_radius;
                    imuEast = (novatel.lon-imu.lon)*D2R * (earth_radius *
cos(novatel.lat*D2R));
                   imuDown = novatel.alt - imu.alt;
                   imuRms = sqrt(imuNorth*imuNorth+imuEast*imuEast);
                }
            }
             else if (currentState == DEBUG_STATE) /* run off IMU only */
             {
                    initialized = FALSE; /* filter is not initialized */
                    gettimeofday(&t0,&z);
                    while (getImuData(&imu, &imuInfo, kFilter, novatel, novatelInfo) != 0)
                      pthread_testcancel();

                    /* write results */
                    kFilter.time  = imu.time ;
                    kFilter.lat   = imu.lat;
                    kFilter.lon   = imu.lon;
                    kFilter.alt   = imu.alt;
                    kFilter.roll  = imu.roll;
                    kFilter.pitch = imu.pitch;
                    kFilter.yaw   = imu.yaw;
                    kFilter.velN  = imu.velN;
                    kFilter.velE  = imu.velE;
                    kFilter.velD  = imu.velD;
                    kFilter.positionRms = 99.99;

             /* calculate yaw rate */
                    if(timePrev != 0.0)
                            kFilter.yawRate = (kFilter.yaw - yawPrev) /
                                            (kFilter.time - timePrev);
                    else
                            kFilter.yawRate = 0.0;
```

```
                        timePrev = kFilter.time;
                        yawPrev = kFilter.yaw;
                        /* calculate loop rate */
                        gettimeofday(&t1,&z);
                        hertz = 1.0 / ((double)(t1.tv_sec - t0.tv_sec)
                                        + (double)(t1.tv_usec - t0.tv_usec)/1.0e6);
                }
                else
                {
                        initialized = FALSE; /* filter is not initialized */
                        timePrev=0.0;
                        yawPrev=0.0;
                        sleep(1); /* just wait */
                }
        }
        return NULL;
}

void interruptHandler(int signo)
{
        desiredState = SHUTDOWN_STATE;
}

int main(int argc, char *argv[])
{
        int i, useConsole=FALSE;
        pthread_attr_t attr;
        struct sched_param p;
        JAUGSmessage_t rxMsg, txMsg;
        emergency_t emergencyData;
        status_t statusData;
        port_t gpsCorrectionsPort;
        FILE *fp;
        FILE *imuCFGfp;
        char line[100], c, d;
        struct _offset
        {
                double x;
                double y;
                double z;
        } imuOffset, novatelOffset;
        double lat, lon ;
        float alt, roll, pitch, yaw, time = 0.0 ;
        float velX, velY, velZ, velN, velE, velD ;
        double sinPsi, cosPsi, sinTheta, cosTheta, sinPhi, cosPhi ;
        double xvec[3], yvec[3], zvec[3] ;
        positionGeographicPosition_t positionData;

        if (argc == 2)
        {
                /* check for console switch */
```

```
                for (i=0; i<strlen(argv[1]); i++)
                        if (argv[1][i] == 'c')
                                useConsole = TRUE;
                if (useConsole != TRUE)
                {
                        printf("Invalid switch %s\n",argv[1]);
                        exit(1);
                }
        }
        else if (argc > 2)
        {
                printf("Usage: %s [-c]\n",argv[0]);
                exit(1);
        }

        /* initialize com port to gps */
        gpsCorrectionsPort = initPort(1,19200,"8N1");

        /* read in user input attitude wref to MAPS */
        if ((imuCFGfp = fopen("imu.cfg","r")) != NULL)
    {
                while ( fgets(line,100,imuCFGfp) != NULL)
          {
                        if(line[0] == 'r')
                    sscanf(line,"%c %d %lf %lf %lf",&d,&USE,&refRoll,&refPitch,&refYaw);
                        if(line[0] == 'c')
                    sscanf(line,"%c %d",&d,&TIME_COARSE);
                        if(line[0] == 'b')
                    sscanf(line,"%c %lf %lf %lf %lf %lf
%lf",&d,&biasX,&biasY,&biasZ,&driftX,&driftY,&driftZ);
                        if(line[0] == 'g')
                    sscanf(line,"%c %d %lf %lf %lf %lf %lf
%lf",&d,&WITHHOLD_GPS,&time1start,&time1stop,&time2start,&time2stop,&time3start,&ti
me3stop);
                        if(line[0] == 'h')
                    sscanf(line,"%c %d %lf %lf
%lf",&d,&HARDCODE_GPS,&HClat,&HClon,&HCalt);
          }
                fclose(imuCFGfp);
    }

        /* read in sensor offsets from config file */
        if ((fp = fopen("pos.cfg","r")) != NULL)
        {
                while ( fgets(line,100,fp) != NULL)
                {
                        if(line[0] == 'r')
                        {
                                sscanf(line,"%c %lf %lf %f",&c,&ref.lat,&ref.lon,&ref.alt);
                        }
                        else if (line[0] == 'm')
```

```
                {
                sscanf(line,"%c %lf %lf %lf",&c,&imuOffset.x,
                                &imuOffset.y,&imuOffset.z);
                }
                else if (line[0] == 'a')
                {
                        sscanf(line,"%c %lf %lf %lf",&c,&novatelOffset.x,
                                        &novatelOffset.y,&novatelOffset.z);
                }
        }
        fclose(fp);

        gps_to_inu[0] = imuOffset.x - novatelOffset.x ;
        gps_to_inu[1] = imuOffset.y - novatelOffset.y ;
        gps_to_inu[2] = imuOffset.z - novatelOffset.z ;
        inu_to_cp[0]  = -imuOffset.x ;
        inu_to_cp[1]  = -imuOffset.y ;
        inu_to_cp[2]  = -imuOffset.z ;
        gps_to_cp[0]  = -novatelOffset.x ;
        gps_to_cp[1]  = -novatelOffset.y ;
        gps_to_cp[2]  = -novatelOffset.z ;
}
else
{
        printf("ERROR:  Configuration file not found\n");
        exit(1);
}
/* intialize communications with the MRS */
while(openComm(MY_COMPONENT_ID,&myNodeId,&mySubsystemId) == -1)
        sleep(1);
/* set up handler for interrupt signal */
signal(SIGINT,&interruptHandler);
/* initial state */
currentState = STANDBY_STATE;
desiredState = READY_STATE;
/* start main thread(s) */
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);
pthread_attr_setschedpolicy(&attr,SCHED_FIFO);
p.sched_priority = sched_get_priority_max(SCHED_FIFO);
pthread_attr_setschedparam(&attr,&p);
pthread_attr_setinheritsched(&attr,PTHREAD_EXPLICIT_SCHED);
pthread_create(&posThreadId, &attr, posThread, NULL);
pthread_attr_destroy(&attr);
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);
pthread_attr_setschedpolicy(&attr,SCHED_FIFO);
p.sched_priority = sched_get_priority_max(SCHED_FIFO);
pthread_attr_setschedparam(&attr,&p);
pthread_attr_setinheritsched(&attr,PTHREAD_EXPLICIT_SCHED);
pthread_create(&readGpsThreadId, &attr, readGpsThread, NULL);
```

```
	pthread_attr_destroy(&attr);

	/* start thread to update component state */
	pthread_attr_init(&attr);
	pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);
	pthread_attr_setschedpolicy(&attr,SCHED_FIFO);
	p.sched_priority = sched_get_priority_min(SCHED_FIFO);
	pthread_attr_setschedparam(&attr,&p);
	pthread_attr_setinheritsched(&attr,PTHREAD_EXPLICIT_SCHED);
	pthread_create(&updateStateThreadId, &attr, updateStateThread, NULL);
	pthread_attr_destroy(&attr);

	if (useConsole)
	{
		/* start thread for console */
		pthread_attr_init(&attr);
		pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_DETACHED);
		pthread_attr_setschedpolicy(&attr,SCHED_FIFO);
		p.sched_priority = sched_get_priority_min(SCHED_FIFO);
		pthread_attr_setschedparam(&attr,&p);
		pthread_attr_setinheritsched(&attr,PTHREAD_EXPLICIT_SCHED);
		pthread_create(&consoleThreadId, &attr, (void *)consoleThread, NULL);
		pthread_attr_destroy(&attr);
	}

/* set up scheduling for main process */
p.sched_priority = sched_get_priority_max(SCHED_FIFO);
sched_setscheduler(getpid(),SCHED_FIFO,&p);
	while (currentState != SHUTDOWN_STATE)
	{
		if ((recvMessage(MY_COMPONENT_ID,&rxMsg)) != -1)
		{
			switch (rxMsg.messageId)
			{
				/* core componet messages */
				case (MY_COMPONENT_ID*256 + SHUTDOWN_MSG):
					desiredState = SHUTDOWN_STATE;
					break;
				case (MY_COMPONENT_ID*256 + STANDBY_MSG):
					desiredState = STANDBY_STATE;
					break;
				case (MY_COMPONENT_ID*256 + RESUME_MSG):
					desiredState = READY_STATE;
			/* Open Data File */
			 if (filesave == 0)
			 {   KFfp = fopen("../pos/data/KF.dat","w");
			    IMUfp = fopen("../pos/data/IMU.dat","w");
			    GPSfp = fopen("../pos/data/GPS.dat","w");
			    ERRfp = fopen("../pos/data/ERROR.dat","w");
			    filesave = 1;
			    break;
```

```
                        }
                    break;
                        case (MY_COMPONENT_ID*256 + RESET_MSG):
                            desiredState = READY_STATE;
                        fclose(KFfp);
                        fclose(IMUfp);
                        fclose(GPSfp);
                        fclose(ERRfp);
                        filesave = 2;
                            break;
                        case (MY_COMPONENT_ID*256 + EMERGENCY_MSG):
                            rxEmergency(&rxMsg,&emergencyData);
                            if (emergencyData.emergencyCode.asField.stop)
                                desiredState = EMERGENCY_STATE;
                            break;
                        case (MY_COMPONENT_ID*256 +
CLEAR_EMERGENCY_MSG):
                            rxClearEmergency(&rxMsg,&emergencyData);
                            if (emergencyData.emergencyCode.asField.stop)
                                desiredState = STANDBY_STATE;
                            break;
                        case (MY_COMPONENT_ID*256 +
QUERY_STATUS_MSG):
                            txMsg.properties.asField.expMsgFlag = 0;
                            txMsg.properties.asField.srvConFlag = 0;
                            txMsg.properties.asField.ackNak = 0;
                            txMsg.properties.asField.priority = 6;
                            txMsg.reserved = 0;
                            txMsg.provSubsystemId = mySubsystemId;
                            txMsg.reqCompId = MY_COMPONENT_ID;
                            txMsg.reqSubsystemId = mySubsystemId;
                            txMsg.sequenceNumber = 0;

                            statusData.primaryStatus.asField.primaryStatusCode =
currentState;
                            statusData.primaryStatus.asField.available = 0;
                            statusData.secondaryStatus.asDoubleWord = 0;
                            txStatus(&txMsg, &statusData);
                            break;
                        /* other component messages */
                        case POSITION_QUERY_GEOGRAPHIC_POSITION:
                            /* use kFilter data */
                            time  = kFilter.time ;
                            lat   = kFilter.lat ;
                            lon   = kFilter.lon ;
                            alt   = kFilter.alt ;
                            roll  = kFilter.roll ;
                            pitch = kFilter.pitch ;
                            yaw   = kFilter.yaw ;
                            velN  =  kFilter.velN ;
                            velE  =  kFilter.velE ;
```

```
velD  =  kFilter.velD ;

/* convert velocity from NED to vehicle XYZ */
sinPsi   = sin(yaw) ;
cosPsi   = cos(yaw) ;
sinTheta = sin(pitch) ;
cosTheta = cos(pitch) ;
sinPhi   = sin(roll) ;
cosPhi   = cos(roll) ;

xvec[0] =  cosPsi*cosTheta ;
xvec[1] =  cosPsi*sinTheta*sinPhi - sinPsi*cosPhi ;
xvec[2] =  cosPsi*sinTheta*cosPhi + sinPsi*sinPhi ;
yvec[0] =  sinPsi*cosTheta ;
yvec[1] =  sinPsi*sinTheta*sinPhi + cosPsi*cosPhi ;
yvec[2] =  sinPsi*sinTheta*cosPhi - cosPsi*sinPhi ;
zvec[0] = -sinTheta ;
zvec[1] =  cosTheta*sinPhi ;
zvec[2] =  cosTheta*cosPhi ;

velX = (float)(velN*xvec[0] + velE*yvec[0] +
velD*zvec[0]) ;

velY = (float)(velN*xvec[1] + velE*yvec[1] +
velD*zvec[1]) ;

velZ = (float)(velN*xvec[2] + velE*yvec[2] +
velD*zvec[2]) ;


while (yaw > PI)
          yaw -= 2.0*PI;
while (yaw < -PI)
          yaw += 2.0*PI;

positionData.latitude = lat;
positionData.longitude = lon;
positionData.elevation = alt;
positionData.positionRMS = kFilter.positionRms;
positionData.roll = roll;
positionData.pitch = pitch;
positionData.yaw = yaw;
positionData.attitudeRMS = 0;
positionData.velocityX = velX;
positionData.velocityY = velY;
positionData.velocityZ = velZ;
positionData.velocityRMS = 0.0;
positionData.rollRate = 0.0;
positionData.pitchRate = 0.0;
positionData.yawRate = kFilter.yawRate;
positionData.omegaRMS = 0.0;
positionData.timeStamp = time;

txPositionGeographicPosition(&txMsg, &positionData);
```

```
                                        break;

                                case 9826: /* gps corrections */
                                        /* send data to gps corrections port */
                                        write(gpsCorrectionsPort,rxMsg.data,rxMsg.dataSize);
                                        break;
                        }
                }
        }
        closeComm(MY_COMPONENT_ID,myNodeId,mySubsystemId);
        /* cancel all threads */
   if ((posThreadId = pthread_cancel(posThreadId)) != 0)
      perror("pthread_cancel");
   if ((readGpsThreadId = pthread_cancel(readGpsThreadId)) != 0)
      perror("pthread_cancel");
   if ((updateStateThreadId = pthread_cancel(updateStateThreadId)) != 0)
      perror("pthread_cancel");
        printf("component %d is exiting\n",MY_COMPONENT_ID);
        return 0;
}

/*---------------------------*/
/*   pos.h                   */
/*---------------------------*/
#ifndef __pos_h
#define __pos_h
#define MY_COMPONENT_ID POSITION

typedef struct _position
{
        double lat;
        double lon;
        float alt;
        float positionRms;
        float roll;
        float pitch;
        float yaw;
        float attitudeRms;
        float velN;
        float velE;
        float velD;
        float velocityRms;
        float rollRate;
        float pitchRate;
float yawRate;
        float attitudeRateRms;
        double time;
} position_t;

typedef struct _imuInfo
{
```

```
        float alignTime;
        int errorCount;
        int status[2];
} imuInfo_t;

typedef struct _novatelInfo
{
        char status;
        int newDataFlag;
} novatelInfo_t;

typedef struct _kFilterInfo
{
        char status;
} kFilterInfo_t;

#ifndef MAIN_PROGRAM
extern pthread_t consoleThreadId;
extern pthread_t posThreadId, readGpsThreadId, updateStateThreadId;
extern unsigned char myNodeId, mySubsystemId;
extern unsigned char desiredState, currentState;
extern position_t novatel, imu, kFilter, ref;
extern imuInfo_t imuInfo;
extern novatelInfo_t novatelInfo;
extern double hertz;
extern int setClockCount;
extern double dt1, dt2;
extern double missionTime;
extern int filesave;
extern int USE;
extern double refRoll,refPitch,refYaw;
extern double biasX,biasY,biasZ,driftX,driftY,driftZ;
extern int TIME_COARSE;
extern double driftNorth,driftEast,driftDown,driftRms;
extern double imuNorth,imuEast,imuDown,imuRms;
extern int STATIC_TEST;
extern double driftRoll,driftPitch,driftYaw;
extern double startroll,startpitch,startyaw;
extern double gps_to_inu[3], inu_to_cp[3], gps_to_cp[3];
extern int NAV_UPDATE;

#endif
#endif


/*-------------------------------------------------------------------------------*/
/* library     : imuLib.c                                                        */
/* Version     : SBC-1.0                                                         */
/* Programmer  : Rommel Mandapat                                                 */
/* Contractor  : Center for Intelligent Machines and Robotics                    */
/*             : University of Florida                                           */
```

```
/* Project     : Autonomous Vehicle System                                    */
/*------------------------------------------------------------------------------*/
/* Date        : 04/13/2001 original creation                                 */
/* Last Update : 06/07/2001                                                   */
/*------------------------------------------------------------------------------*/
/* This library contains the following functions defined in imuLib.h:     */
/*------------------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <sys/time.h>
#include "pos.h"
#include "imuLib.h"

#include "portOpsLib.h"

int getImuData(position_t *data, imuInfo_t *imuInfo, position_t kFilter, position_t novatel,
novatelInfo_t novatelInfo)
{
   int mode = 0;
   GeoStruct GeoData;
   imuInfo_t Info;
   int FAKE_IMU = FALSE;

   imuGetGeodeticData(&GeoData, &Info, kFilter, novatel, novatelInfo);

   if (GeoData.good_data)
   {
      data->time  = GeoData.time_stamp;
      data->lat   = GeoData.latitude;
      data->lon   = GeoData.longitude;
      data->alt   = GeoData.altitude;
      data->roll  = GeoData.roll;
      data->pitch = GeoData.pitch;
      data->yaw   = GeoData.azimuth;
      data->velN  = GeoData.vel_n;
      data->velE  = GeoData.vel_e;
      data->velD  = GeoData.vel_d;


      if ((FAKE_IMU) && (missionTime > 10))
      {  data->lat = 29.646428;
         data->lon = -82.349362;
         data->alt = 12.34;
         data->roll = 1.38*D2R*1;
         data->pitch = -2.03*D2R*1;
         data->yaw = -3.0*D2R*1;
         data->velN = 0.0;
         data->velE = 0.0;
```

```
        data->velD = 0.0;
      }

      imuInfo->status[0] = Info.status[0];
      imuInfo->status[1] = Info.status[1];
      imuInfo->alignTime = Info.alignTime;

      if (Info.status[0] == ALIGN_MODE)
        mode = 1;
      if (Info.status[0] == NAVIGATION)
        mode = 0;

     return mode;
    }
    else
    {
      imuInfo->errorCount++;
      return -1;
    }
}


#define BUF_SIZE 255

int receive(ImuStruct *ImuData)
{
  port_t imuDataPort;
  FILE *fp;
  char buf[BUF_SIZE], header[6];

  double omeX,omeY,omeZ,accX,accY,accZ;

/* initialize port */
  imuDataPort = initPort(2,19200,"8N1");
  flushBuf(imuDataPort,BOTH_BUF);

  fp=fdopen(imuDataPort,"r");

  while(1)
  {
    fgets(buf,BUF_SIZE,fp);

    if (strncmp(buf,"$IMU",4) == 0)
    {

sscanf(buf,"%4s,%lf,%lf,%lf,%lf,%lf,%lf",header,&omeX,&omeY,&omeZ,&accX,&accY,&acc
Z);

       ImuData->omegaX = omeX;
       ImuData->omegaY = omeY;
       ImuData->omegaZ = omeZ;
```

```
        ImuData->accelX = accX;
        ImuData->accelY = accY;
        ImuData->accelZ = accZ;

        close(imuDataPort);

        return 1;
     }
    else
        close(imuDataPort);
        return 0;
  }
}

void imuGetGeodeticData(GeoStruct *GeoData, imuInfo_t *Info, position_t kFilter, position_t
novatel, novatelInfo_t novatelInfo)
{

/* Calibration Coeff : Biases and Drifts */
/*  double biasX = 5.385437e-3;  5.385437e-3;
  double biasY = -5.674602e-3; -5.674602e-3;
  double biasZ = -8.352242e-3;  -1.735224e-2;
  double driftX = -1.37536e-5;  -1.37546e-5;
  double driftY = 6.959467e-6;  6.959467e-6;
  double driftZ = 1.038893e-5; 1.038893e-5;
*/
  ImuStruct ImuData;
  double IMU_LATENCY = 0.0e-6;

  double omegaX,omegaY,omegaZ;
  double accX,accY,accZ;

  static int OKalign = 0;
  static double rolli = 0.0;
  static double pitchi = 0.0;
  static double yawi = 0.0;

  double S_b[3];
  double w_b[3];
  double X_plus[6];
  double lat_plus,lon_plus,H_plus;
  static double R_B2V_plus[3][3];
  static double X[6];
  static double lat,lon,H;
  static double R_B2V[3][3];

  static double roll, pitch, yaw;
  static double r,p,y;
  int j,k;
  double R_n,R_m;
  static int navcount = 0;
```

```
 int new_gps_data;

 struct timeval t;
 struct timezone z;
 int goodData;
 int clock_sec_of_day;
 int SEC_IN_DAY = 86400;

 int USE_KF_DATA = FALSE;

 imuInfo_t iInfo;

/* Receive Data Message from IMU */
 goodData = receive(&ImuData);
 GeoData->good_data = goodData;

/* synchronize IMU time w/ SBC clock */
 if (goodData == TRUE)
 { gettimeofday(&t, &z);
   clock_sec_of_day = t.tv_sec % SEC_IN_DAY;
   GeoData->time_stamp = (double)clock_sec_of_day
                  + (double)t.tv_usec/1.0e6 - IMU_LATENCY ;

   /* Add manual calibration biases and drifts */
   /* Rotate CS : vehicle Y (+ left), vehicle Z (+ up) */
   omegaX = ImuData.omegaX + driftX;
   omegaY = -ImuData.omegaY + driftY;
   omegaZ = -ImuData.omegaZ + driftZ;
   accX = ImuData.accelX + biasX;
   accY = -ImuData.accelY + biasY;
   accZ = -ImuData.accelZ + biasZ;

   if (OKalign == FALSE)  /* Aligning  */
   {
     GeoData->latitude = novatel.lat;
     GeoData->longitude = novatel.lon;
     GeoData->altitude = novatel.alt;
     GeoData->vel_n = -11.11;
     GeoData->vel_e = -11.11;
     GeoData->vel_d = -11.11;
     GeoData->roll = -11.11*D2R;
     GeoData->pitch = -11.11*D2R;
     GeoData->azimuth = -11.11*D2R;
     GeoData->good_data = TRUE;

     OKalign =
startAlign(omegaX,omegaY,omegaZ,accX,accY,accZ,&rolli,&pitchi,&yawi,novatel,novatelInfo,
&iInfo);

     Info->status[0] = ALIGN_MODE;
     Info->status[1] = iInfo.status[1];
```

```
    Info->alignTime = iInfo.alignTime;

    if (OKalign == TRUE)
    {  GeoData->roll = rolli;
       GeoData->pitch = pitchi;
       GeoData->azimuth = yawi;
       Info->status[1] = ALIGNED;
       Info->status[0] = NAVIGATION;
       Info->alignTime = 0.0;
    }
  }

  else  /* Align complete & Begin Navigation */
  {
    Info->alignTime = 0.0;
    Info->status[0] = NAVIGATION;
    Info->status[1] = ALIGNED;

    R_n = earth_radius / sqrt(1-e2*sin(lat)*sin(lat)); /* normal_radius */
    R_m = R_n * (1 - e2) / (1-e2*sin(lat)*sin(lat));   /* meridian_radius */

    if (navcount == 0)
    {
      B2V(rolli,pitchi,yawi,R_B2V);

      lat = novatel.lat*D2R;
      lon = novatel.lon*D2R;
      H = (double)novatel.alt;

      X[0] = 0;
      X[1] = 0;
      X[2] = 0;
      X[3] = -R_n*e2*sin(lat)*cos(lat);
      X[4] = 0;
      X[5] = R_n*(1-e2*(sin(lat)*sin(lat)))+H;
    }

    S_b[0] = accX;
    S_b[1] = accY;
    S_b[2] = accZ;
    w_b[0] = omegaX;
    w_b[1] = omegaY;
    w_b[2] = omegaZ;

  /* CALL NAVIGATION SOLUTION FUNCTION */

navsol(X_plus,&lat_plus,&lon_plus,&H_plus,R_B2V_plus,X,lat,lon,H,R_B2V,S_b,w_b);
      ++navcount;

      getangle(R_B2V_plus,&roll,&pitch,&yaw);
      r = roll;
```

```
    p = pitch;
    y = yaw;

   /* Update states */
    lat = lat_plus;
    lon = lon_plus;
    H = H_plus;

    new_gps_data = novatelInfo.newDataFlag;

    if (USE_KF_DATA)
    { if ((navcount%4000)==0)
      {
         r = kFilter.roll;
         p = -kFilter.pitch;
         y = -kFilter.yaw;
        /* r = rolli;
         p = pitchi;
         y = yawi;
       */  lat = novatel.lat*D2R;
         lon = novatel.lon*D2R;
         H = (double)novatel.alt;
         X_plus[0] = kFilter.velN;
         X_plus[1] = -kFilter.velE;
         X_plus[2] = -kFilter.velD;
         NAV_UPDATE = TRUE;
         printf("Update! ");
         }
     }

    B2V(r,p,y,R_B2V_plus);

    for (j=0;j<6;++j)
            X[j] = X_plus[j];
    for (j=0;j<3;++j)
       for (k=0;k<3;++k)
              R_B2V[j][k] = R_B2V_plus[j][k];


/* write IMU data to shared memory */
    GeoData->latitude = lat*R2D;
    GeoData->longitude = lon*R2D;
    GeoData->altitude = H;
    GeoData->vel_n = X[0];   /* north velocity */
    GeoData->vel_e = -X[1];  /* east velocity, negative of vel_w */
    GeoData->vel_d = -X[2];  /* down velocity, negative of vel_u */
    GeoData->roll = r;       /* same roll */
    GeoData->pitch = -p;     /* negative of pitch */
    GeoData->azimuth = -y;  /* negative of yaw */
    GeoData->good_data = TRUE ;
  }
```

```
  }
}


int startAlign (double Omega_X, double Omega_Y, double Omega_Z, double Accel_X, double
Accel_Y, double Accel_Z, double *rolli, double *pitchi, double *yawi, position_t novatel,
novatelInfo_t novatelInfo, imuInfo_t *iInfo)
{
        static double tot_Omega_X;
        static double tot_Omega_Y;
        static double tot_Omega_Z;
        static double tot_Accel_X;
        static double tot_Accel_Y;
        static double tot_Accel_Z;
        double ave_Omega_X = 0.0;
        double ave_Omega_Y = 0.0;
        double ave_Omega_Z = 0.0;
        double ave_Acc_X = 0.0;
        double ave_Acc_Y = 0.0;
        double ave_Acc_Z = 0.0;
        double omega_data_rms = 0.0;
        double accel_data_rms = 0.0;
        int DATA_USE = 0;
        int DATA_MODE = 0;
      static double refLat;      /* Reference Latitude (GPS) */

/***  Static Condition Parameters  ***/
      int STATIC_SEC = 3;  /* min consecutive sec bef collect static data */
      int STATIC_START = STATIC_SEC*freq;  /* min consecutive static data points bef collect
static data */
      int MOVE_SEC = 5;  /* no. of consecutive sec before align reset */
      int MOVE_LIMIT = MOVE_SEC*freq; /* no. of cons data bef align reset */
      static int MOVE_COUNTER = 0;
      static int STATIC_COUNTER = 0;

/***  Coarse Alignment Parameters  ***/
        static int ALIGN_STATUS = 0;
  /* No. of consecutive static data points to begin coarse align */
        int STATIC_COARSE ;
      static double TIME_LEFT;
        double roll, pitch, yaw;

    if ((ALIGN_STATUS == RESET)&&(STATIC_COUNTER==0))
    {
      if (novatelInfo.newDataFlag == 1)
      refLat = novatel.lat*D2R;
    }

    STATIC_COARSE = (int)(TIME_COARSE*freq) ;
    if (STATIC_COUNTER == 0)
    {  tot_Omega_X = 0.0;
```

```
        tot_Omega_Y = 0.0;
        tot_Omega_Z = 0.0;
        tot_Accel_X = 0.0;
        tot_Accel_Y = 0.0;
        tot_Accel_Z = 0.0;
      }
      TIME_LEFT = (double)((STATIC_COARSE - STATIC_COUNTER +
STATIC_START)/freq);

      iInfo->alignTime = TIME_LEFT;
      iInfo->status[1] = COARSE_ALIGN;

      if ((STATIC_COUNTER-STATIC_START) <= STATIC_COARSE)
      {
        omega_data_rms =
sqrt(Omega_X*Omega_X+Omega_Y*Omega_Y+Omega_Z*Omega_Z);
        accel_data_rms = sqrt(Accel_X*Accel_X+Accel_Y*Accel_Y+Accel_Z*Accel_Z);

        DATA_USE = static_test(omega_data_rms,accel_data_rms,DATA_MODE);

            if (DATA_USE == TRUE)
            {  ++ STATIC_COUNTER;
              if (STATIC_COUNTER>STATIC_START)
              {
                tot_Omega_X = tot_Omega_X + Omega_X;
                tot_Omega_Y = tot_Omega_Y + Omega_Y;
                tot_Omega_Z = tot_Omega_Z + Omega_Z;
                tot_Accel_X = tot_Accel_X + Accel_X;
                tot_Accel_Y = tot_Accel_Y + Accel_Y;
                tot_Accel_Z = tot_Accel_Z + Accel_Z;
          }
        }
            else
            {  ++ MOVE_COUNTER;

              if (MOVE_COUNTER > MOVE_LIMIT)
              {   ALIGN_STATUS = RESET ;
                    STATIC_COUNTER = 0;
               MOVE_COUNTER = 0;
              }
            }
    return 0;
    }

    else
    {
            ave_Omega_X = tot_Omega_X / (STATIC_COUNTER-STATIC_START);
            ave_Omega_Y = tot_Omega_Y / (STATIC_COUNTER-STATIC_START);
            ave_Omega_Z = tot_Omega_Z / (STATIC_COUNTER-STATIC_START);
            ave_Acc_X = tot_Accel_X / (STATIC_COUNTER-STATIC_START);
            ave_Acc_Y = tot_Accel_Y / (STATIC_COUNTER-STATIC_START);
```

```
            ave_Acc_Z = tot_Accel_Z / (STATIC_COUNTER-STATIC_START);

        coarseAlign(ave_Omega_X, ave_Omega_Y, ave_Omega_Z, ave_Acc_X, ave_Acc_Y,
ave_Acc_Z, &roll, &pitch, &yaw, refLat);

        *rolli = roll;
        *pitchi = pitch;
        *yawi = yaw;

        STATIC_COUNTER = 0;

        return 1;
      }
}


int static_test(double omega_data_rms, double accel_data_rms, int DATA_MODE)
{
    double omega_rms_thresh = 0.005 ; /* Angular Velocity RMS threshhold value */
          double accel_rms_lowlimit = 9.73 ; /* 9.73 Acceleration RMS lower limit value */
          double accel_rms_uprlimit = 9.85 ; /* 9.85 Acceleration RMS upper limit value */

          if (omega_data_rms < omega_rms_thresh)
          {   if ((accel_data_rms > accel_rms_lowlimit)&(accel_data_rms < accel_rms_uprlimit))
              {   DATA_MODE = STATIC ;
                            return 1;
       }
       else
       {  DATA_MODE = MOVING;
                        return 0;
                     }
          }
          else
          {   DATA_MODE = MOVING;
                    return 0;
    }
}


void coarseAlign(double Omega_X, double Omega_Y, double Omega_Z, double Acc_X, double
Acc_Y, double Acc_Z, double *roll, double *pitch, double *yaw, double refLat)
{
   double fn[3];
   double wn[3];
   double vn[3];
   double fb[3];
   double wb[3];
   double vb[3];
   double M[3][3];
   double Minv[3][3];
   double Q[3][3];
```

```
double R_B2V_th[3][3];
double R_B2V[3][3];

double r=0;
double p=0;
double y=0;

init_matrix(R_B2V_th);
init_matrix(R_B2V);
init_matrix(M);

/* Specific Force (gravity) vector in Nav Coord */
fn[0] = -0.0146;
fn[1] = 0;
fn[2] = -grav;

/* Earth Rotation Vector in Nav Coord */
wn[0] = wie*cos(refLat);
wn[1] = 0;
wn[2] = wie*sin(refLat);

cross(vn, fn, wn);

M[0][0] = fn[0];
M[0][1] = fn[1];
M[0][2] = fn[2];
M[1][0] = wn[0];
M[1][1] = wn[1];
M[1][2] = wn[2];
M[2][0] = vn[0];
M[2][1] = vn[1];
M[2][2] = vn[2];

invert_matrix(Minv, M);

fb[0] = Acc_X  ;
fb[1] = Acc_Y  ;
fb[2] = Acc_Z  ;

wb[0] = Omega_X  ;
wb[1] = Omega_Y  ;
wb[2] = Omega_Z  ;

cross(vb,fb,wb);

Q[0][0] = fb[0];
Q[0][1] = fb[1];
Q[0][2] = fb[2];
Q[1][0] = wb[0];
Q[1][1] = wb[1];
```

```
  Q[1][2] = wb[2];
  Q[2][0] = vb[0];
  Q[2][1] = vb[1];
  Q[2][2] = vb[2];

  matmult(R_B2V, Minv, 3, 3, Q, 3, 3);

  getangle(R_B2V, &r, &p, &y);

  *roll = r;
  *pitch = p;
  *yaw = y;

  if (USE)
  {
  printf("Using Ref Att !");
    *roll = refRoll*D2R;
    *pitch = -refPitch*D2R;
    *yaw = -refYaw*D2R;
  }
}


void getangle(double R_B2V[row][col], double *r, double *p, double *y)
{
    *r = atan2(R_B2V[2][1],R_B2V[2][2]);
    *p = asin(-R_B2V[2][0]);
    *y = atan2(R_B2V[1][0],R_B2V[0][0]);
}


void B2V(double r, double p, double y, double R_B2V[row][col])
{
        R_B2V[0][0] =  cos(p)*cos(y);
        R_B2V[1][0] =  cos(p)*sin(y);
        R_B2V[2][0] = -sin(p);
        R_B2V[0][1] = -cos(r)*sin(y)+sin(r)*sin(p)*cos(y);
        R_B2V[1][1] =  cos(r)*cos(y)+sin(r)*sin(p)*sin(y);
        R_B2V[2][1] =  sin(r)*cos(p);
        R_B2V[0][2] =  sin(r)*sin(y)+cos(r)*sin(p)*cos(y);
        R_B2V[1][2] = -sin(r)*cos(y)+cos(r)*sin(p)*sin(y);
        R_B2V[2][2] =  cos(r)*cos(p);
}


void matmult(double ans[row][col], double matrix1[row][col], int row1, int col1, double
matrix2[row][col], int row2, int col2)
{
        int i,j,k ;
        double temp=0;
```

```
        if (col1 != row2)
          exit;

        init_matrix(ans);

        for (i=0;i<row1;++i)
        {       for (j=0;j<col2;++j)
                {       for (k=0;k<col1;++k)
                        {       temp = matrix1[i][k]*matrix2[k][j] ;
                                ans[i][j] = ans[i][j] + temp ;
                        }
                }
        }
}

void vecmult(double ans2[row], double matrix1[row][col], int row1, int col1, double
vector1[row], int row2)
{
        int i,k ;
        double temp;

        init_vector(ans2);

        if (col1 != row2)
          exit;

        for (i=0;i<row1;++i)
        {       for (k=0;k<col1;++k)
                {       temp = matrix1[i][k]*vector1[k] ;
                        ans2[i] = ans2[i] + temp ;
                }
        }
}

void matrix_add(double ans[row][col], double matrix1[row][col], int row1, int col1, double
matrix2[row][col], int row2, int col2)
{
        int i,j ;

        init_matrix(ans);

        for (i=0;i<row1;++i)
        {       for (j=0;j<col1;++j)
                {       ans[i][j] = matrix1[i][j] + matrix2[i][j] ;
                }
        }
}

void matrix_subtract(double ans[row][col], double matrix1[row][col], int row1, int col1, double
matrix2[row][col], int row2, int col2)
{
```

```
	int i,j ;

	init_matrix(ans);

	for (i=0;i<row1;++i)
	{	for (j=0;j<col1;++j)
		{	ans[i][j] = matrix1[i][j] - matrix2[i][j] ;
		}
	}
}

void vector_add(double ans2[row], double vector1[row], int row1, double vector2[row], int row2)
{
	int i;

	init_vector(ans2);

	for (i=0;i<row1;++i)
	{	ans2[i] = vector1[i] + vector2[i] ;
	}
}

void vector_subtract(double ans2[row], double vector1[row], int row1, double vector2[row], int
row2)
{
	int i;

	init_vector(ans2);

	for (i=0;i<row1;++i)
	{	ans2[i] = vector1[i] - vector2[i] ;
	}
}

void cross(double ans2[row], double vector1[row], double vector2[row])
{
	init_vector(ans2);

	ans2[0] = vector1[1]*vector2[2] - vector1[2]*vector2[1];
	ans2[1] = vector1[2]*vector2[0] - vector1[0]*vector2[2];
	ans2[2] = vector1[0]*vector2[1] - vector1[1]*vector2[0];
}


void transpose(double ans[row][col], double matrix[row][col])
{
	int i,j ;

	init_matrix(ans);

	for (i=0;i<row;++i)
```

```
        {       for (j=0;j<col;++j)
                {       ans[j][i] = matrix[i][j] ;
                }
        }
}

void identity_matrix(double result[row][col])
{
        int i,j ;

        if (row != col)
          exit;

        init_matrix(result);

        for (i=0;i<row;++i)
        {       for (j=0;j<col;++j)
                        result[i][j] = 0;
           result[i][i] = 1;
        }

}

void invert_matrix(double result[row][col], double matrix[row][col])
{
        int i,j,k,p ;
        double temp_matrix[row][2*row];
        double Id[row][row] ;
        double temp_vec = 0;
        double divisor[row];
        double val=0;
        double max;
        int rownum;
        double div;

      if (row != col)
           exit;

/*      init_matrix(temp_matrix);*/
        init_matrix(result);
        init_vector(divisor);
        identity_matrix(Id);

        p = row;

        /* form augmented matrix (original matrix + identity matrix, n x 2n) */
        for (i=0;i<p;++i)
        {       for (j=0;j<p;++j)
           {   temp_matrix[i][j] = matrix[i][j];
               temp_matrix[i][j+p] = Id[i][j];
                }
```

```
        }

   /* Gaussian reduction */

   for (i=0 ; i<col ; ++i)
   {   max = 0.0 ;
       rownum = i ;

       for (j=rownum ; j<row ; ++j)
       {   val = fabs(temp_matrix[j][i]) ;
           if (val>max)
           {   max = val ;
               rownum = j ;
           }
       }

  /* swap row 'rownum' with row 'i' */
       if (rownum != i)
       {   for (j=0; j<(2*col); ++j)
                       {
                                   temp_vec = temp_matrix[i][j] ;
             temp_matrix[i][j] = temp_matrix[rownum][j] ;
             temp_matrix[rownum][j] = temp_vec ;
                       }
       }

  /* if the pivot value is close to zero, then we have a problem
       if (valuenear(temp_matrix[i][i], 0.0, 0.00001))
       {
          exit ;
       }
*/
              div = temp_matrix[i][i];
  /* make the pivot value equal to one */
       for (j=0; j<(2*col); ++j)
                       {
                 temp_matrix[i][j] = temp_matrix[i][j]/div ;
                       }

       for (j=0 ; j<row ; ++j)
       {
          if (j == i)
            continue ;

          val = temp_matrix[j][i] ;
          for (k=0; k<(2*col); ++k)
                                  {
                                    temp_matrix[j][k] = temp_matrix[j][k] - val * temp_matrix[i][k] ;
                                  }
             }
          }
```

```
		for (i=0; i<p; ++i)
		{
			for (j=0; j<(2*p); ++j)
			{
				result[i][j] = temp_matrix[i][j+p];
			}
		}
}

void print_matrix(double ans[row][col])
{
    int i,j;

		for (i=0;i<row;++i)
		{	printf("\n");
			for (j=0;j<col;++j)
				printf(" %.8f ",ans[i][j]) ;
		}
		printf("\n");
}

void print_vector(double ans2[row])
{
	int i;

		for (i=0;i<row;++i)
			printf("\n %.8f ",ans2[i]) ;
		printf("\n");

}

void init_matrix(double matrix1[row][col])
{
    int i,j;

		for (i=0;i<row;++i)
			for (j=0;j<col;++j)
			{   matrix1[i][j] = 0 ;
			}
}

void init_vector(double vector[row])
{
    int i;

		for (i=0;i<row;++i)
		{   vector[i] = 0 ;
		}
}
```

```
void navsol(double X_plus[6],double *lat_plus,double *lon_plus,double *H_plus,double
R_B2V_plus[3][3],double X[6],double lat,double lon,double H,double R_B2V[3][3],double
S_b[3],double w_b[3])
{

/* conversion constants  */

/* variables */
  double R_n = 0;
  double R_m = 0;
  double In[3][3];
  double In4[4][4];
  double Ia[6][6];
  double Zn[3][3];
  double lon_i = 0;
  double V_n,V_w,V_u;
  double P_n,P_w,P_u;
  double R_C2V[3][3];
  double R_I2V[3][3];
  double R_E2V[3][3];
  double gn,gw,gu;
  double w_ie2v[3];
  double omega_IE2V[3][3];
  double w_ev2v[3];
  double omega_EV2V[3][3];
  double w_iv2v[3];
  double omega_IV2V[3][3];
  double A1[3][3];
  double A2[3][3];
  double A[6][6];
  double B[6][9];
  double U[9];
  double R_V2B[3][3];
  double V1[3];
  double Vtemp[3];
  double w_ib2b[3];
  double q1,q2,q3,q4;
  double q[4];
  double q_plus[4];
  int n,m;
  double tempM[3][3];
  double tempM2[3][3];
  double phi,theta,psi;
  double R,P,Y;

/* calculate earth radius components */
  R_n = earth_radius / sqrt(1-e2*sin(lat)*sin(lat));           /* normal_radius */
  R_m = R_n * (1 - e2) / (1-e2*sin(lat)*sin(lat));      /* meridian_radius */

    identity_matrix(In);
```

```
    for (m=0;m<4;++m)
    {  for (n=0;n<4;++n)
         In4[m][n] = 0.0;
       In4[m][m] = 1.0;
    }

    for (m=0;m<6;++m)
    {  for (n=0;n<6;++n)
         Ia[m][n] = 0.0;
       Ia[m][m] = 1.0;
    }

    init_matrix(Zn);

    lon_i = lon + wie/dt;

/* Initial Rotation matrix for inertial (I) to navigation (V) */
    I2V(R_I2V,lat,lon_i);

    V_n=X[0];
    V_w=X[1];
    V_u=X[2];

    X[3] = -R_n*e2*sin(lat)*cos(lat);
    X[4] =  0;
    X[5] = R_n*(1-e2*(sin(lat)*sin(lat)))+H;

    P_n=X[3];
    P_w=X[4];
    P_u=X[5];

/* Initial Rotation matrix for LCV (C) to navigation (V) */
    C2V(R_ C2V,P_n,P_w,P_u);

/* Initial Rotation matrix for inertial (I) to navigation (V) */
    I2V(R_E2V,lat,lon);

/* compute gravity vector */

   gn = - wie*wie*(earth_radius+H)*sin(2*lat)/2;
   gw = 0.0;
   gu = grav + wie*wie*(earth_radius+H)*(1+cos(2*lat))/2;


/*** setup up Skewed symmetric matrices  ***/

    w_ie2v[0] = wie*cos(lat);
    w_ie2v[1] = 0;
    w_ie2v[2] = wie*sin(lat);
```

```
    skewsymm(omega_IE2V,w_ie2v);

    w_ev2v[0] = -V_w/(R_n+H);
    w_ev2v[1] = V_n/(R_m+H);
    w_ev2v[2] = (-tan(lat)*V_w)/(R_n+H);

    skewsymm(omega_EV2V,w_ev2v);

    w_iv2v[0] = w_ie2v[0] + w_ev2v[0];
    w_iv2v[1] = w_ie2v[1] + w_ev2v[1];
    w_iv2v[2] = w_ie2v[2] + w_ev2v[2];

    skewsymm(omega_IV2V,w_iv2v);

/********** Define the A matrix **********/

    matrix_add(A1,omega_IV2V,3,3,omega_IE2V,3,3);

    matmult(A2,omega_IE2V,3,3,omega_IE2V,3,3);

    for (m=0;m<6;++m)
      for (n=0;n<6;++n)
        A[m][n] = 0.0;

    A[0][0] = -A1[0][0];
    A[0][1] = -A1[0][1];
    A[0][2] = -A1[0][2];
    A[0][3] = -A2[0][0];
    A[0][4] = -A2[0][1];
    A[0][5] = -A2[0][2];
    A[1][0] = -A1[1][0];
    A[1][1] = -A1[1][1];
    A[1][2] = -A1[1][2];
    A[1][3] = -A2[1][0];
    A[1][4] = -A2[1][1];
    A[1][5] = -A2[1][2];
    A[2][0] = -A1[2][0];
    A[2][1] = -A1[2][1];
    A[2][2] = -A1[2][2];
    A[2][3] = -A2[2][0];
    A[2][4] = -A2[2][1];
    A[2][5] = -A2[2][2];
    A[3][0] = 1;
    A[3][1] = 0;
    A[3][2] = 0;
    A[3][3] = -omega_EV2V[0][0];
    A[3][4] = -omega_EV2V[0][1];
    A[3][5] = -omega_EV2V[0][2];
    A[4][0] = 0;
    A[4][1] = 1;
    A[4][2] = 0;
```

```
    A[4][3] = -omega_EV2V[1][0];
    A[4][4] = -omega_EV2V[1][1];
    A[4][5] = -omega_EV2V[1][2];
    A[5][0] = 0;
    A[5][1] = 0;
    A[5][2] = 1;
    A[5][3] = -omega_EV2V[2][0];
    A[5][4] = -omega_EV2V[2][1];
    A[5][5] = -omega_EV2V[2][2];


/********** Define the B matrix ***********/

    for(m=0;m<6;++m)
      for(n=0;n<9;++n)
            B[m][n] = 0.0;

    B[0][0] = R_C2V[0][0];
    B[0][1] = R_C2V[0][1];
    B[0][2] = R_C2V[0][2];
    B[0][3] = R_ E2V[0][0];
    B[0][4] = R_E2V[0][1];
    B[0][5] = R_E2V[0][2];
    B[0][6] = R_B2V[0][0];
    B[0][7] = R_B2V[0][1];
    B[0][8] = R_B2V[0][2];
    B[1][0] = R_C2V[1][0];
    B[1][1] = R_C2V[1][1];
    B[1][2] = R_C2V[1][2];
    B[1][3] = R_E2V[1][0];
    B[1][4] = R_E2V[1][1];
    B[1][5] = R_E2V[1][2];
    B[1][6] = R_B2V[1][0];
    B[1][7] = R_B2V[1][1];
    B[1][8] = R_B2V[1][2];
    B[2][0] = R_C2V[2][0];
    B[2][1] = R_C2V[2][1];
    B[2][2] = R_C2V[2][2];
    B[2][3] = R_E2V[2][0];
    B[2][4] = R_E2V[2][1];
    B[2][5] = R_E2V[2][2];
    B[2][6] = R_B2V[2][0];
    B[2][7] = R_B2V[2][1];
    B[2][8] = R_B2V[2][2];

    U[0] = 0.0018;
    U[1] = 0;
    U[2] = -9.81;
    U[3] = 0;
    U[4] = 0;
    U[5] = 0;
```

```
   U[6] = S_b[0];
   U[7] = S_b[1];
   U[8] = S_b[2];

/*  State equation:  X_plus = expAdt*X + expAdt*B*U */

   RK45nav(X_plus,dt,A,X,B,U);

   V_n = X_plus[0];
   V_w = X_plus[1];
   V_u = X_plus[2];
   P_n = X_plus[3];
   P_w = X_plus[4];
   P_u = X_plus[5];

 /* transport rate in navigation frame */

 /* Height at time, t=t+dt */
   *H_plus = H + V_u*dt;

 /* latitude at time, t=t+dt */
   *lat_plus = lat + (V_n/(R_m+H))*(dt);

 /* longitude at time, t=t+dt */
   *lon_plus = lon - ((V_w/cos(lat))/(R_n+H))*(dt);



/******** compute rotational changes  **********/

/* Gyro Output Eq:  w_ib2b = w_b-R_B2V'*w_ie2v;  */

   transpose(R_V2B,R_B2V);

   vector_add(Vtemp,w_ie2v,3,w_ev2v,3);

   vecmult(V1,R_V2B,3,3,Vtemp,3);

   vector_subtract(w_ib2b,w_b,3,V1,3);

 /* initialize quaternions */
   q1 = 0.5*sqrt(1+R_B2V[0][0]+R_B2V[1][1]+R_B2V[2][2]);

 if (q1 == 0)
 {  q2 = 0;
    q3 = 0;
    q4 = 1;
 }
 else
 {  getangle(R_B2V,&phi,&theta,&psi);
```

```
    if (psi<0)
    {
      q1 = -q1;
    }
    q2 = (R_B2V[2][1]-R_B2V[1][2])/(4*q1);
    q3 = (R_B2V[0][2]-R_B2V[2][0])/(4*q1);
    q4 = (R_B2V[1][0]-R_B2V[0][1])/(4*q1);
  }

  q[0] = q1;
  q[1] = q2;
  q[2] = q3;
  q[3] = q4;

  RK45quat(q_plus,dt,q,w_ib2b);

  QUAT(R_B2V_plus,q_plus);

 /* Ensure Orthogonality */
  getangle(R_B2V_plus,&R,&P,&Y);
  B2V(R,P,Y,R_B2V_plus);

 /* Orthogonality Check */
  transpose(tempM,R_B2V_plus);
  matmult(tempM2,R_B2V_plus,3,3,tempM,3,3);
}


void C2V(double R_C2V[3][3],double P_n,double P_w,double P_u)
{
          double P = 0;

          P = sqrt(P_n*P_n+P_u*P_u);

    if (P==0)
      init_matrix(R_C2V);
    else
            R_C2V[0][0] = P_u/P;
            R_C2V[0][1] = 0;
            R_C2V[0][2] = P_n/P;
            R_C2V[1][0] = 0;
            R_C2V[1][1] = 1;
            R_C2V[1][2] = 0;
            R_C2V[2][0] = -P_n/P;
            R_C2V[2][1] = 0;
            R_C2V[2][2] = P_u/P;
}



/*-------------------------------------------------------------------------*/
```

```
/* Procedure   : E2I()                                          */
/* Version     : SBC-1.0                                        */
/* Programmer  : Rommel Mandapat                                */
/* Contractor  : Center for Intelligent Machines and Robotics   */
/*             : University of Florida                          */
/* Project     : Autonomous Vehicle System                      */
/*--------------------------------------------------------------*/
/* Date        : 04/13/2001 original creation                   */
/* Last Update : 06/07/2001                                     */
/*--------------------------------------------------------------*/
/* This function calculates a rotation matrix from ECEF to ECI  */
/*--------------------------------------------------------------*/
void E2I(double R_E2I[3][3],double dtime)
{
        R_E2I[0][0] =  cos(wie)*dtime;
        R_E2I[0][1] = -sin(wie)*dtime;
        R_E2I[0][2] =  0.0;
        R_E2I[1][0] =  sin(wie)*dtime;
        R_E2I[1][1] =  cos(wie)*dtime;
        R_E2I[1][2] =  0.0;
        R_E2I[2][0] =  0.0;
        R_E2I[2][1] =  0.0;
        R_E2I[2][2] =  1.0;
}

void I2V(double R_I2V[3][3],double lat,double lon)
{
        R_I2V[0][0] = -cos(lon)*sin(lat);
        R_I2V[0][1] = -sin(lon)*sin(lat);
        R_I2V[0][2] =  cos(lat);
        R_I2V[1][0] =  sin(lon);
        R_I2V[1][1] =  -cos(lon);
        R_I2V[1][2] =  0;
        R_I2V[2][0] = cos(lon)*cos(lat);
        R_I2V[2][1] = sin(lon)*cos(lat);
        R_I2V[2][2] = sin(lat);
}


void QUAT(double quat[3][3],double q[4])
{
        double q1,q2,q3,q4;

        q1 = q[0];
    q2 = q[1];
    q3 = q[2];
    q4 = q[3];

        quat[0][0] = q1*q1+q2*q2-q3*q3-q4*q4;
        quat[0][1] = 2*(q2*q3-q1*q4);
        quat[0][2] = 2*(q2*q4+q1*q3);
```

```
        quat[1][0] = 2*(q2*q3+q1*q4);
        quat[1][1] = q1*q1-q2*q2+q3*q3-q4*q4;
        quat[1][2] = 2*(q3*q4-q1*q2);
        quat[2][0] = 2*(q2*q4-q1*q3);
        quat[2][1] = 2*(q3*q4+q1*q2);
        quat[2][2] = q1*q1-q2*q2-q3*q3+q4*q4;
}


void skewsymm(double SKEWMAT[3][3],double VEC[3])
{
   SKEWMAT[0][0] = 0;
   SKEWMAT[0][1] = -VEC[2];
   SKEWMAT[0][2] = VEC[1];
   SKEWMAT[1][0] = VEC[2];
   SKEWMAT[1][1] = 0;
   SKEWMAT[1][2] = -VEC[0];
   SKEWMAT[2][0] = -VEC[1];
   SKEWMAT[2][1] = VEC[0];
   SKEWMAT[2][2] = 0;
}


void DEQnav(double X_dot[6],double A[6][6],double X[6],double B[6][9],double U[9])
{
   double X1[6];
   double X2[6];
   int jj,kk;
   double temp;

   for (jj=0;jj<6;++jj)
   {   X1[jj] = 0;
      X2[jj] = 0;
   }

/*  State equation:  X_dot = AX + BU */

   for (jj=0;jj<6;++jj)
   {      for (kk=0;kk<6;++kk)
          {   temp = A[jj][kk]*X[kk] ;
                  X1[jj] = X1[jj] + temp ;
          }
   }

   for (jj=0;jj<6;++jj)
   {      for (kk=0;kk<9;++kk)
          {   temp = B[jj][kk]*U[kk] ;
             X2[jj] = X2[jj] + temp ;
          }
   }
```

```
   for (jj=0;jj<6;++jj)
          X_dot[jj] = X1[jj] + X2[jj] ;
}


void RK45nav(double X_plus[6],double dtime,double A[6][6],double X[6],double
B[6][9],double U[9])
{
   double tn = 0.0;
   double tn_temp;
   double K1[6];
   double K2[6];
   double K3[6];
   double K4[6];
   double Vx[6];
   double x_temp[6];
   int ii;

   DEQnav(Vx,A,X,B,U);
   for (ii=0;ii<6;++ii)
          K1[ii] = dtime*Vx[ii];

   tn_temp = tn + 0.5*dtime;
   for (ii=0;ii<6;++ii)
     x_temp[ii] = X[ii] + 0.5*K1[ii];


   DEQnav(Vx,A,x_temp,B,U);
   for (ii=0;ii<6;++ii)
     K2[ii] = dtime*Vx[ii];

   tn_temp = tn + 0.5*dtime;
   for (ii=0;ii<6;++ii)
     x_temp[ii] = X[ii] + 0.5*K2[ii];
   DEQnav(Vx,A,x_temp,B,U);
   for (ii=0;ii<6;++ii)
     K3[ii] = dtime*Vx[ii];

   tn_temp = tn + dtime;
   for (ii=0;ii<6;++ii)
     x_temp[ii] = X[ii] + K3[ii];
   DEQnav(Vx,A,x_temp,B,U);
   for (ii=0;ii<6;++ii)
     K4[ii] = dtime*Vx[ii];

   for (ii=0;ii<6;++ii)
     X_plus[ii] = X[ii] + (1.0/6.0)*(K1[ii]+2*K2[ii]+2*K3[ii]+K4[ii]);
}


void DEQquat(double q_dot[4], double q[4], double w[3])
```

```
{
   double w_xb, w_yb, w_zb;
   double Q1, Q2, Q3, Q4;

   w_xb = w[0];
   w_yb = w[1];
   w_zb = w[2];
   Q1 = q[0];
   Q2 = q[1];
   Q3 = q[2];
   Q4 = q[3];

/*  Quaternion Propagation */

   q_dot[0] = -0.5*(Q2*w_xb + Q3*w_yb + Q4*w_zb);
   q_dot[1] = 0.5*(Q1*w_xb - Q4*w_yb + Q3*w_zb);
   q_dot[2] = 0.5*(Q4*w_xb + Q1*w_yb - Q2*w_zb);
   q_dot[3] = -0.5*(Q3*w_xb - Q2*w_yb - Q1*w_zb);
}


void RK45quat(double q_plus[4],double dtime,double q[4],double w[3])
{
   double tn = 0.0;
   double tn_temp;
   double K1[4];
   double K2[4];
   double K3[4];
   double K4[4];
   double Qx[4];
   double q_temp[4];
   int ii;

   DEQquat(Qx,q,w);
   for (ii=0;ii<4;++ii)
         K1[ii] = dtime*Qx[ii];
   tn_temp = tn + 0.5*dtime;
   for (ii=0;ii<4;++ii)
     q_temp[ii] = q[ii] + 0.5*K1[ii];
   DEQquat(Qx,q_temp,w);
   for (ii=0;ii<4;++ii)
     K2[ii] = dtime*Qx[ii];

   tn_temp = tn + 0.5*dtime;
   for (ii=0;ii<4;++ii)
     q_temp[ii] = q[ii] + 0.5*K2[ii];
   DEQquat(Qx,q_temp,w);
   for (ii=0;ii<4;++ii)
     K3[ii] = dtime*Qx[ii];

   tn_temp = tn + dtime;
```

```
    for (ii=0;ii<4;++ii)
      q_temp[ii] = q[ii] + K3[ii];
    DEQquat(Qx,q_temp,w);
    for (ii=0;ii<4;++ii)
      K4[ii] = dtime*Qx[ii];

    for (ii=0;ii<4;++ii)
      q_plus[ii] = q[ii] + (1.0/6.0)*(K1[ii]+2*K2[ii]+2*K3[ii]+K4[ii]);
}


/*-------------------------------------------------------------------------*/
/* library     : imuLib.h                                                  */
/* Version     : SBC-1.0                                                   */
/*-------------------------------------------------------------------------*/
/* Date        : 04/20/2001 original creation                              */
/* Last Update : 06/26/2001                                                */
/*-------------------------------------------------------------------------*/

#ifndef __imuLib_h
#define __imuLib_h
#define PRINT       1
#define TRUE        1
#define FALSE       0
#define row 3  /* maximum no of rows */
#define col 3  /* maximum no of columns */
#define s 8

  /* system parameters */
#define  freq    12.5     /* data frequency in hertz */
#define  dt      1/freq       /* data time step */
#define  pi      3.14159
#define  D2R     pi/180
#define  R2D     180/pi

  /* navigation model parameters */
#define  earth_radius  6378137.0          /* radius of earth in meters */
#define  e2          0.00669437999013  /* square of the earth eccentricity */
#define  WGS_grav_0   -9.7803267714  /* gravity 0 */
#define  WGS_grav_1   -0.00193185138639       /* gravity 1 */
#define  grav        -9.801   /* gravitational constant + up direction*/
#define  wie          7.292115e-5  /* earth rot rate, + ccw about north axis */

/* IMU status */
#define START_UP              0
#define POS_UPDATE_REQUEST      1
#define ALIGN_MODE            2
#define NAVIGATION            3
#define ZUPT_MODE             4
#define SHUTDOWN_COMPLETE       5
```

```
/* ALIGN status */
#define RESET                0
#define ALIGNED              1
#define COARSE_ALIGN         2
#define FINE_ALIGN           3
#define POS_UPDATE_IN_PROGRESS  4
#define ZUPT_IN_PROGRESS     5

/* ALIGN DATA status */
#define STATIC               1
#define MOVING               0

typedef  struct {
    double latitude    ;
    double longitude   ;
    float  altitude    ;
    float  vel_n       ;
    float  vel_e       ;
    float  vel_d       ;
    float  roll        ;
    float  pitch       ;
    float  azimuth     ;
    double time_stamp  ;
    int    imu_status  ;
    int    align_status ;
    double align_time  ;
    int    good_data   ;
    }  GeoStruct ;

 typedef  struct {
    char     header[4] ;
    double   omegaX    ;
    double   omegaY    ;
    double   omegaZ    ;
    double   accelX    ;
    double   accelY    ;
    double   accelZ    ;
    }  ImuStruct ;

 typedef  struct {
    double   roll      ;
    double   pitch     ;
    double   yaw       ;
    }  AttitudeStruct  ;

/* Function Prototypes */
int getImuData(position_t *data, imuInfo_t *imuInfo, position_t kFilter, position_t novatel,
novatelInfo_t novatelInfo);
int receive(ImuStruct *ImuData);
void imuGetGeodeticData(GeoStruct *GeoData, imuInfo_t *Info, position_t kFilter, position_t
novatel, novatelInfo_t novatelInfo);
```

```
int startAlign (double Omega_X, double Omega_Y, double Omega_Z, double Accel_X, double
Accel_Y, double Accel_Z, double *rolli, double *pitchi, double *yawi, position_t novatel,
novatelInfo_t novatelInfo, imuInfo_t *iInfo);
int static_test(double omega_data_rms, double accel_data_rms, int DATA_MODE);
void coarseAlign(double Omega_X, double Omega_Y, double Omega_Z, double Acc_X, double
Acc_Y, double Acc_Z, double *roll, double *pitch, double *yaw, double refLat);
void getangle(double R_B2V[row][col], double *r, double *p, double *y);
void B2V(double r, double p, double y, double R_B2V[row][col]);
void matmult(double ans[row][col], double matrix1[row][col], int row1, int col1, double
matrix2[row][col], int row2, int col2);
void vecmult(double ans2[row], double matrix1[row][col], int row1, int col1, double
vector1[row], int row2);
void matrix_add(double ans[row][col], double matrix1[row][col], int row1, int col1, double
matrix2[row][col], int row2, int col2);
void matrix_subtract(double ans[row][col], double matrix1[row][col], int row1, int col1, double
matrix2[row][col], int row2, int col2);
void vector_add(double ans2[row], double vector1[row], int row1, double vector2[row], int
row2);
void vector_subtract(double ans2[row], double vector1[row], int row1, double vector2[row],int
row2);
void cross(double ans2[row], double vector1[row], double vector2[row]);
void transpose(double ans[row][col], double matrix[row][col]);
void identity_matrix(double result[row][col]);
void invert_matrix(double result[row][col], double matrix[row][col]);
void print_matrix(double ans[row][col]) ;
void print_vector(double ans2[row]) ;
void init_matrix(double matrix1[row][col]);
void init_vector(double vector[row]);
void C2V(double R_C2V[3][3],double P_n,double P_w,double P_u);
void E2I(double R_E2I[3][3],double dtime);
void I2V(double R_I2V[3][3],double lat,double lon);
void QUAT(double quat[3][3],double q[4]);
void skewsymm(double SKEWMAT[3][3],double VEC[3]);
void DEQquat(double q_dot[4], double q[4], double w[3]);
void RK45quat(double q_plus[4],double dtime,double q[4],double w[3]);
void DEQnav(double X_dot[6],double A[6][6],double X[6],double B[6][9],double  U[9]);
void RK45nav(double X_plus[6],double dtime,double A[6][6],double X[6],double B[6][9],
double U[9]);
void navsol(double X_plus[6],double *lat_plus,double *lon_plus,double *H_plus, double
R_B2V_plus[3][3], double X[6], double lat, double lon, double  H, double R_B2V[3][3], double
S_b[3], double w_b[3]);

#endif
#ifndef __filter_h
#define __filter_h

/*  filterLib.h  */

#define SQR(x)          ((x) * (x))
#define QUEUE_SIZE  40
```

```
/* INU model parameters */
#define earth_radius        6378137.0         /* meters               */
#define eccentricity_sqr    0.00669437999013
#define earth_rate          7.292115e-5       /* radians per second   */
#define WGS_grav_0          9.7803267714      /* meters per sec sqr   */
#define WGS_grav_1          0.00193185138639
#define inu_k1_const        0.0           /* 1.0e-4 baro damp not used*/
#define inu_k2_const        0.0           /* 1.0e-6 baro damp not used*/

/* IMU filter parameters */
#define     number_time_store   30
#define     number_states       9
#define     number_meas         3
#define     u_pos_meas          5.0e-1          /* meters              */
#define     u_dpos_meas         5.0e-1          /* meters              */
#define     e_pos_error         2.0e+1          /* meters              */
#define     e_alt_error         2.0e+1          /* meters              */
#define     e_vel_error         1.0e+0          /* meters per second   */
#define     e_tlt_error         1.0e-4          /* radians             */
#define     e_hdg_error         1.0e-3          /* radians             */
#define     grav_const          9.801           /* m/sq sec            */
#define     u_hori_grav         210.0           /* arc-sec             */
#define     u_vert_grav         1000.0          /* micro-g's           */
#define     u_accel_SF          300.0           /* parts per million   */
#define     u_accel_MA          200.0           /* arc-sec             */
#define     u_gyro_rw           0.125           /* degree per root hr  */
#define     skip_factor         8.0
#define     hori_grav_error     u_hori_grav * ars_to_rad * grav_const
#define     accel_SF_error      u_accel_SF / 1.0e+6
#define     accel_MA_error      u_accel_MA * ars_to_rad

/* INU filter states */
#define inu_lat_err         0
#define inu_long_err        1
#define inu_alt_err         2
#define inu_lat_rate_err    3
#define inu_long_rate_err   4
#define inu_alt_rate_err    5
#define inu_north_tilt      6
#define inu_east_tilt       7
#define inu_down_tilt       8

#define NORTH  0
#define EAST   1
#define DOWN   2

#define X      0
#define Y      1
#define Z      2
```

```
typedef struct inu_type {
        double time;
        double latitude;
        double longitude;
        double altitude;
        double roll;
        double pitch;
        double yaw;
        double vel_north;
        double vel_east;
        double vel_down;
        double roll_rate;
        double pitch_rate;
        double yaw_rate;
} INU_DATA, *INU_DATA_PTR;

typedef struct queue_struct {
        INU_DATA inu[QUEUE_SIZE] ;
        int ptr ;
} QUEUE, *QUEUE_PTR;

typedef struct gps_type {
        double time;
        double latitude;
        double longitude;
        double altitude;
} GPS_DATA, *GPS_DATA_PTR;

typedef struct nc_type {
        double time;
        double delta_update_time;
        double latitude;
        double longitude;
        double altitude;
        double roll;
        double pitch;
        double yaw;
        double vel_east;
        double vel_north;
        double vel_down;
        double rms;
} NC_DATA, *NC_DATA_PTR;

typedef struct _NCqueue {
        NC_DATA nc ;
        struct _NCqueue *next ;
} NCQUEUE, *NCQUEUE_PTR ;

/* function prototypes */
int INU_initialize (int *pos_update,INU_DATA inu, GPS_DATA gps, NC_DATA_PTR nc,
  QUEUE_PTR q, INU_DATA_PTR inu_past, GPS_DATA_PTR gps_past,
```

```
  double Phi[number_states][number_states],
  double F[number_states][number_states],
  double P[number_states][number_states], double Q[number_states],
  double x[number_states], double H[number_meas][number_states],
  double R[number_meas][number_meas]);

#endif

/*-----------------------------------------------------------*/
/* POS.cfg                                                   */
/* POS configuration file                                    */
/*    includes the sensor offsets to the control point       */
/*    and reference positions                                */
/*-----------------------------------------------------------*/
1
# Reference latitude, longtitude, and altitude
# UF
# Honeywell HG1700AG11 IMU / Novatel RT-20 GPS Pos System

# CIMAR LAB (MEB 131)
# r 29.646412 -82.349352 12.36

# Bandshell Base Point 1 (Original Ashtech)
# r 29.6472687 -82.35443972 15.000000

# Bandshell Base Point 2 (Novatel)
r 29.64723355 -82.3544477 14.965000

# Tyndall
# r 29.968349 -85.4729078 0.000000

# -- Sensor offsets from Vehicle Coordinate System --

# IMU offsets in meters (from CP, Control Point)
# from MULE ashtech gps antenna (CP)
m -0.571000 0.140000 0.782000

# MULE center rear axle
# m 0.386000 0.140000 0.782000

# ASV
# m -2.019300 0.152400 0.000000

# AMRADS
# m -0.630 0.494 0.000000


# GPS (Novatel) offsets in meters (from CP)
# from MULE ashtech gps antenna (CP)
a 0.402000 0.000000 0.000000
```

```
# MULE center rear axle
# a 0.703000 0.000000 0.000000

# ASV
# a 0.000000 0.000000 -1.574800

# AMRADS
# a 0.0 0.0 -0.837


/*-------------------------------------------------------------------------*/
/* imu.cfg                                                                 */
/* IMU configuration file                                                  */
/*    includes the settings for sensor biases and testing parameters       */
/*    to aid in tuning the kalman filter                                   */
/*-------------------------------------------------------------------------*/
1
# Reference Attitude (Roll, Pitch, Yaw)


# Use this to input reference attitude angles
# from known source such as MAPS or compass
# r (USE=1 to use ref & USE=0 to not use) (Ref Roll) (Ref Pitch) (Ref Yaw)
# (all ref angles in degrees)
r 0 1.23 -2.66 -1.71

# Input Alignment Parameters
# c (Coarse Alignment Time in full seconds)
c 60

# Input IMU Biases and Drifts
# b (Bias X) (Bias Y) (Bias Z) (Drift X) (Drift Y) (Drift Z)
# biases in m/sec^2,  drifts in rad/sec
b 5.385437e-3 -5.674602e-3 -8.352242e-3 -1.37536e-5 6.959467e-6 1.038893e-5

# WITHHOLD GPS
# Input Time of Temporary Loss of GPS
# g (Withhold GPS = 1 -> True, 0 -> False) (Time 1 Start) (Time 1 Stop)
# (Time 2 Start) (Time 2 Stop) (Time 3 Start) (Time 3 Stop)
# all times are referenced to mission time and are in seconds
g 0 60.0 65.0 120.0 130.0 180.0 200.0


# Hardcode GPS Position
# CIMAR LAB
h 1 29.646418 -82.349352 12.34
# BANDSHELL
# h 0 29.6472687 -82.35443972 15.00
```

```
/*****************************************************************/
/*   Program:  imudatacom.c                                      */
/*   This program takes raw data from the IMU, reads it in using DMA in MS-DOS */
/*   through the Seamac ACB-104 serial communications card, performs error  */
/*   checks on the data, averages it from 100 Hz down to 12.5 Hz, and outputs it  */
/*   serially (RS232)  in an IMU message format                  */
/*****************************************************************/
#include    <bios.h>
#include    <dos.h>
#include    <stdio.h>
#include    <stdlib.h>
#include    <math.h>
#include    <string.h>
#include    <conio.h>
#include    <ctype.h>
#include    "serial.c"
#include    "serial.h"
#include    "sstypes.h"
#include    "acb_rtl.h"

#define pi 3.14159
#define N 100
#define R2D 180/pi
#define F2M 0.3048

#define ESC 27          /*keyboard character to terminate program*/
#define BUFFER_LENGTH      45
int main(void);
void cls(void);

BYTE   RxBuffer[1024]="Empty buffer\0";
#define RX_BUFFER_LENGTH    100
LPBYTE          fpBuffer;
unsigned char  counter;
unsigned char  counter2;

WORD wRxFrameCount = 0;
WORD wTxFrameCount = 0;
WORD wErrorCount = 0;
/*****************************************************************/
/*****************************************************************/
int main (void)
{   DWORD dwTemp = 0;
    BYTE c = 0;
    WORD    wTemp = 0, i;
    int ii = 0;
    int jj = 0;
    int bufcount=0;
    int goodcount = 0;
    int totalcount = 0;
```

```
int diff = 0;
FILE *ofp;
int startcounter;
int time = 0;
int starttime = 0;
int endtime = 0;
int timediff = 0;
int minutes = 0;
double seconds = 0.0;

unsigned char buff[N] ;
double fieldvar[24];
double value[6];
double d_theta[N];
double old_dthetax = 0.0;
double add_dthetax = 0.0;
double total_thetax = 0.0;
double total_thetaxp = 0.0;
double omega_x = 0.0;
double old_dthetay = 0.0;
double add_dthetay = 0.0;
double total_thetay = 0.0;
double total_thetayp = 0.0;
double omega_y = 0.0;
double old_dthetaz = 0.0;
double add_dthetaz = 0.0;
double thetaz = 0.0;
double ave_dthetaz = 0.0;
double total_thetaz = 0.0;
double total_thetazp = 0.0;
double omega_z = 0.0;
double d_vel[N];
double old_dvelx = 0.0;
double add_dvelx = 0.0;
double accel_x = 0.0;
double old_dvely = 0.0;
double add_dvely = 0.0;
double accel_y = 0.0;
double old_dvelz = 0.0;
double add_dvelz = 0.0;
double accel_z = 0.0;
double time_interval = 0.01;    /* IMU message period, 1/100 Hz */
int count = 0;
double maxval = 4294967295.0;   /* max value of data field */
double midval = 2147483647.0;   /* min value of data field */
double dthetx = 0.0;
double dthety = 0.0;
double dthetz = 0.0;
double dvelx = 0.0;
double dvely = 0.0;
double dvelz = 0.0;
```

```
    double vel_RMS = 0.0;

    /* Data Threshhold Values */
    double thresh_theta = 0.2;  /* 0.2, threshhold value in delta radians */
    double thresh_vel = 4.0;    /* 4.0, threshhold value in delta ft/sec  */
    double diff_theta = 0.04;   /* 0.04, differential threshhold value */
    double diff_vel = 1.0;      /* 1.0, differential threshhold value */
    double RMS_Thresh = 0.02;

    int size = 42;          /* IMU message byte size */
    int m = 0;
    int k = 0;
    int ave_data_rate = 8;   /* 100 Hz / output data rate (12.5 Hz) */
                             /*  8  --> 12.5 Hz */
                             /*  10 --> 10 Hz */
    unsigned long testcount = 0;

    char numStr[100] ;       /* string output for averaged IMU data */

/* Communications parameters */
    int  port     = COM2;
    int  speed    = 19200;
    int  parity   = NO_PARITY;
    int  bits     = 8;
    int  stopbits = 1;

    if (SetSerial(port, speed, parity, bits, stopbits) != 0) {
      fprintf(stderr, "Serial Port setup error, COMM%d.\n", port) ;
      exit( 99 ) ;
    }

    initserial() ;

    for (i=0; i<N; i++)
    {
      d_theta[i] = 0.0;
      d_vel[i] = 0.0;
    }
    system("cls");
    printf("\nHoneywell HG1700AG11 IMU Data Communications Program\n\n");
    printf("-------------------------------------------------------\n");
    /* Verify driver is installed */
    if(ACBVerify())
          {
                printf("\nError: Driver not installed.\n");
                return(1);
          }

    printf("\nDOS DMA Driver present verified....\n");
    printf("\n\nOutputting IMU data at 12.5 Hz\n");
    printf("Type <Esc> to exit.\n");
```

```
ACBStartDriver();        /* Init driver */

startcounter = 0;

      do
      {
            if(_bios_keybrd(_KEYBRD_READY))
            {
              c = (unsigned char)(_bios_keybrd(_KEYBRD_READ));
              if(c != ESC)
              {
              }  /* end if(c != ESC) */
            }  /* end if(_bios_keybrd(_KEYBRD_READY)) */

            if (ACBCheckRxAvailable())
            { //printf("Got it!\n");
              fpBuffer = (LPBYTE)&RxBuffer;
              dwTemp = ACBGetFrame(fpBuffer,1024);

            for (i=0; i<size; i++)
            { buff[i] = *(fpBuffer+i);
              if (buff[i] == 0x0a)
                { if (buff[i+1] == 0x02)
                  { count = 1;
                      ++ m ;
              /* transfer desired datafields into field buffer */
                    for (k=0; k<24; k++)
                      fieldvar[k] = (unsigned long)buff[i+k+18];

              /* compute base 10 value */
                      for (k=0; k<6; k++)
                        value[k] = fieldvar[k*4] + fieldvar[k*4+1]*256 +
fieldvar[k*4+2]*256*256 + fieldvar[k*4+3]*256*256*256;
                  /* Convert Delta Theta and Delta Velocity values */

            /* delta Theta X data */
                  if (value[0] > midval)
                /* negative value */
                        d_theta[m] = -1*(maxval-value[0])*(pow(2,-33));
                      else
                /* positive value */
                        d_theta[m] = value[0]*(pow(2,-33));
                /* Limit to threshhold value / countercheck for bad data */
                      if (fabs(d_theta[m])>thresh_theta)
                        d_theta[m] = old_dthetax;
                      if (fabs(d_theta[m]-old_dthetax)>diff_theta)
                        d_theta[m] = old_dthetax;
                      dthetx = d_theta[m];

            /* delta Theta Y data */
                      if (value[1] > midval)
```

```
    /* negative value */
              d_theta[m] = -1*(maxval-value[1])*(pow(2,-33));
            else
    /* positive value */
              d_theta[m] = value[1]*(pow(2,-33));
    /* Limit to threshhold value / countercheck for bad data */
            if (fabs(d_theta[m])>thresh_theta)
              d_theta[m] = old_dthetay;
            if (fabs(d_theta[m]-old_dthetay)>diff_theta)
              d_theta[m] = old_dthetay;
            dthety = d_theta[m];

  /* delta Theta Z data */
       if (value[2] > midval)
    /* negative value */
              d_theta[m] = -1*(maxval-value[2])*(pow(2,-33));
       else
    /* positive value */
              d_theta[m] = value[2]*(pow(2,-33));
    /* Limit to threshhold value / countercheck for bad data */
            if (fabs(d_theta[m])>thresh_theta)
              d_theta[m] = old_dthetaz;
            if (fabs(d_theta[m]-old_dthetaz)>diff_theta)
              d_theta[m] = old_dthetaz;
            dthetz = d_theta[m];

    /* delta Velocity X data */
            if (value[3] > midval)
              d_vel[m] =   -1*(maxval-value[3])*(pow(2,-27));
            else
              d_vel[m] = value[3]*(pow(2,-27));
            if (fabs(d_vel[m])>thresh_vel)
              d_vel[m] = old_dvelx;
            if (fabs(d_vel[m]-old_dvelx)>diff_vel)
              d_vel[m] = old_dvelx;
            dvelx = d_vel[m];

   /* delta Velocity Y data */
            if (value[4] > midval)
              d_vel[m] =   -1*(maxval-value[4])*(pow(2,-27));
            else
              d_vel[m] = value[4]*(pow(2,-27));
            if (fabs(d_vel[m])>thresh_vel)
              d_vel[m] = old_dvely;
            if (fabs(d_vel[m]-old_dvely)>diff_vel)
              d_vel[m] = old_dvely;
            dvely = d_vel[m];

   /* delta Velocity Z data */
            if (value[5] > midval)
              d_vel[m] =   -1*(maxval-value[5])*(pow(2,-27));
```

```
                else
                  d_vel[m] = value[5]*(pow(2,-27));
                if (fabs(d_vel[m])>thresh_vel)
                  d_vel[m] = old_dvelz;
                if (fabs(d_vel[m]-old_dvelz)>diff_vel)
                  d_vel[m] = old_dvelx;
                dvelz = d_vel[m];

            /* Error Check - Zero Value Data */
                vel_RMS = sqrt(dvelx*dvelx+dvely*dvely+dvelz*dvelz);
                if (vel_RMS < RMS_Thresh)
                {    dvelx = old_dvelx;
                     dvely = old_dvely;
                     dvelz = old_dvelz;
                     dthetx = old_dthetax;
                     dthety = old_dthetay;
                     dthetz = old_dthetaz;
                }

                old_dthetax = dthetx;
                add_dthetax = add_dthetax + old_dthetax;
                old_dthetay = dthety;
                add_dthetay = add_dthetay + old_dthetay;
                old_dthetaz = dthetz;
                add_dthetaz = add_dthetaz + old_dthetaz;
                old_dvelx = dvelx;
                add_dvelx = add_dvelx + old_dvelx;
                old_dvely = dvely;
                add_dvely = add_dvely + old_dvely;
                old_dvelz = dvelz;
                add_dvelz = add_dvelz + old_dvelz;


                if (m>(ave_data_rate-1))
                {
          /* Compute angular velocities from delta thetas */
                  omega_x = add_dthetax / (ave_data_rate*time_interval);
                  omega_y = add_dthetay / (ave_data_rate*time_interval);
                  omega_z = add_dthetaz / (ave_data_rate*time_interval);
                  accel_x = add_dvelx*F2M / (ave_data_rate*time_interval);
                  accel_y = add_dvely*F2M / (ave_data_rate*time_interval);
                  accel_z = add_dvelz*F2M / (ave_data_rate*time_interval);

                 /* total delta thetas for current angle */
                  total_thetax = total_thetaxp +
omega_x*R2D*time_interval*ave_data_rate;
                  total_thetaxp = total_thetax;
                  total_thetay = total_thetayp +
omega_y*R2D*time_interval*ave_data_rate;
                  total_thetayp = total_thetay;
```

```
                                total_thetaz = total_thetazp +
omega_z*R2D*time_interval*ave_data_rate;
                                total_thetazp = total_thetaz;

                                m = 0;
                                add_dthetax = 0.0;
                                add_dthetay = 0.0;
                                add_dthetaz = 0.0;
                                add_dvelx = 0.0;
                                add_dvely = 0.0;
                                add_dvelz = 0.0;
                                }
                              if ((testcount%(ave_data_rate))==0)
                              {

sprintf(numStr,"$IMU,%.8f,%.8f,%.8f,%.6f,%.6f,%.6f\r\n",omega_x,omega_y,omega_z,accel_x,
accel_y,accel_z);
                                SerialString( numStr ) ;
                              }
                              ++ testcount;
                              }
                        }
                     }
                }
}
   while(c != ESC);         /* end do while*/
   printf("count = %d\n",testcount);
   ACBStopDriver();
   closeserial() ;

   return(0);
}


/*********************************************************************
            END OF FILE
 *********************************************************************/
```

LIST OF REFERENCES

Arm93 Armstrong, D.G., "Position and Obstacle Detection Systems for an Outdoor, Land-Based, Autonomous Vehicle," Master's Thesis, University of Florida, 1993.

Arm94 Armstrong, D.G., and Crane, C.D., "Sonar Based Obstacle Avoidance for Outdoor Autonomous Navigation," Proceedings of the Florida Conference on Recent Advances in Robotics, Gainesville, FL, April 1994. 137-140.

Ash93 Ashtech, Ashtech XII GPS Receiver Operating Manual, Ashtech Inc., Sunnyvale CA, 1993.

Bie99 Biezad, D., Integrated Navigation and Guidance Systems. Reston, VA: American Institute of Aeronautics and Astronautics, Inc., 1999.

Bri70 Britting, K and Palsson, Thorgeir., "Self-Alignment Techniques for Strapdown Inertial Navigation Systems with Aircraft Applications," Journal of Aircraft, Vol.7, No.4, 1970, 302-307.

Cha97 Chatfield, Averil B., Fundamentals of High Accuracy Inertial Navigation. Reston, VA: American Institute of Aeronautics and Astronautics, Inc., 1997.

Cla96 Clarke, B., GPS Aviation Applications. New York: McGraw-Hill, 1996.

Com95 Committee on the Future of the Global Positioning System, The Global Positioning System, A Shared National Asset. Washington, D.C.: National Academy Press, 1995.

Dip94 Di Prinzio, M.D. and Tolson, R.H. "Evaluation of GPS Position and Attitude Determination for Automated Rendezvous and Docking Missions," Masters Thesis, Joint Institute for Advancement of Flight Sciences, L, George Washington University, 1994.

Dol Dolan, J., Hampshire, J., Khosla, P., Bhat, K., Diehl, C., and Oliver, S., "Cyber ATV Platform," The Institude of Complex Engineered Systems, Carnegie Mellon University, Pittsburgh, PA.

Hon00 Honeywell Sensor and Guidance Products. Honeywell HG1700 Inertial Measurement Unit Product Description Manual. Minneapolis, MN, 2000.

Hon99 Honeywell Sensor and Guidance Products. Honeywell HG1700AG Inertial Measurement Unit Reference Manual. Minneapolis, MN, 1999.

Hon92 Honeywell Military Avionics Division, H-726 Modular Azimuth Positioning System. St. Petersburg, FL, 1992.

Hut97 Hutchinson, D., "Application of Global Positioning Systems to Autonomous Navigation," Master's Thesis, University of Florida, 1997.

Jar83 Jarvis, R.A., "Growing Polyhedral Obstacles for Planning Collision Free Paths," The Australian Computer Journal, 15(3), 1983, 103-111.

Joc95 Jochem, T., Pomerleau, D., Kumar, B., and Armstrong, J., "PANS: A Portable Navigation Platform," IEEE Symposium on Intelligent Vehicles, Detroit, MI, September 1995.

Joi00 Joint Architecture for Unmanned Ground Systems, "Reference Architecture, Volume II, Version 2.0," Unmanned Ground Vehicles/Systems Joint Project Office AMSAM-DSA-UG, Alabama, September 2000.

Ken Kenny, P., Bidlack, C., Kluge, K., Lee, J., Huber, M., Durfee, E., and Weymouth, T., "Implementation of a Reactive Autonomous Navigation System on an Outdoor Mobile Robot," Artificial Intelligence Laboratory, The University of Michigan, Ann Arbor, Michigan.

Law93 Lawrence, A., Modern Inertial Technology, Navigation, Guidance and Control. New York: Springer-Verlag, 1993.

Log95 Logsdon, T., Understanding the NAVSTAR, GPS, GIS, and IVHS. New York: Van Nostrand Reinhold, 1995.

Mer96 Merhav, S., Aerospace Sensor Systems and Applications. New York: Springer-Verlag, 1996.

Nov01 Novatel Inc. Novatel RT-20 GPS Receiver Product Description Manual. Calgary, Canada, 2001.

Nov98 Novatel Inc. Novatel Beeline GPSCard User Manual. Calgary, Canada, 2001.

Nov97 NovAtel Inc. Novatel Millenium GPSCard Command Descriptions Manual. Calgary, Canada, 1997.

Oco O'Connor, M., Bell, T., Elkaim, G, and Parkinson, B., "Autonomous Steering of Farm Vehicles Using GPS," Stanford University, California.

Ran94a Rankin, A.L. and Crane, C.D. III, "Off-Line Path Planning Using an A* Search Algorithm," Proceedings of the Florida Conference on Recent Advances in Robotics, Gainesville, FL. April 1994, 218-225.

Ran94b Rankin, A.L., Crane, C.D. III, and Armstrong, D.G., "Navigation of an Autonomous Robot Vehicle," ASCE Conference on Robotics for Challenging Environments, Albuquerque, February 1994, 44-51.

Rog96 Rogers, R.M., Wit, J.S., Crane, C.D., and Armstrong, D.G., "Integrated INU/DGPS for Autonomous Vehicle Navigation," Proceedings of IEEE PLANS 96, Atlanta, GA, April, 1996.

Shi92 Shin, D.H., Sanjiv, S., and Lee, J.J., "Explicit Path Tracking by Autonomous Vehicles," Robotica, 10, (1992), 69-87.

Ste95 Stevens, M., Stevens, A. and Durrant-Whyte, H., "Robust Vehicle Navigation," International Symposium on Experimental Roboptics, Stanford, CA, July 1995.

Tit97 Titterton, D.H., Weston, J.L., Strapdown Inertial Navigation Technology. UK: Peter Peregrinus Ltd., 1997.

Upa89 Upadhyay, T.N., Priovolos, G.J. Rhodenhamel, H., Autonomous Integrated GPS/INS Navigation Experiment for OMV: Phase I Feasibility Study, NASA Contractor Report 4267, Reading, MA: Mayflower Communications Company, Inc., 1989.

Wit96 Wit, J., "Integrated Inertial Navigation System and Global Positioning System for the Navigation of an Autonomous Ground Vehicle," Master's Thesis, University of Florida, 1996.

BIOGRAPHICAL SKETCH

The author, Rommel E. Mandapat, was born on June 26, 1968 in Quezon City, Philippines. He received his bachelor's degree in Mechanical Engineering from the University of the Philippines on April of 1991. After which, he passed the Philippine Professional Engineer Licensure Examination in Mechanical Engineering given on November 1991.

From January 1992 to December 1993, he worked for two engineering companies in Japan. In Nippo Denshi, he was a mechanical designer of electronics enclosures and vacuum systems. In Ohtsuka Polytech, he then took a role as automation engineer, working with plastic and rubber automobile parts for Honda and Subaru.

His work experience was further enhanced by serving as Head of Mechanical Engineering for MK Screens, Inc. in Quezon City, Philippines from January 1994 to December 1998. His main role was head design engineer for automation projects in support of manufacturing. In MK, he also acted as Project Engineer for a Steel Mill Transplant project in Seattle, Washington from March 1997 to July 1997.

In December of 1998, he married Emalynn Santos in the Philippines. Soon after they both moved to Gainesville, Florida, where he pursued his Master's degree in Mechanical Engineering at the University of Florida.